UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

VITÓRIA MAINARDI ROSA

Elaboração de contratos de prestação de serviços baseados em blockchain

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Alexandre Carissimi

Porto Alegre
2019

AGRADECIMENTOS

Gostaria de agradecer minha mãe por sempre estar presente, me dar forças para fazer o TCC e ajudar sempre que podia, meu namorado por ter tido paciência e me ajudado tanto com o trabalho quanto com meu bem estar emocional, ao meu pai por ter entendido que eu não poderia estar sempre presente devido ao TCC e ao Carissimi por ter sido um ótimo professor e orientador e pelas várias dicas que me deu ao longo do processo.

RESUMO

A tecnologia *blockchain* surgiu para, dentre outros fatores, facilitar aplicações e operações que precisam de autenticação ou validação. Utilizando um sistema colaborativo e seguro, tais autenticações são possíveis, graças a implementação de uma rede de usuários. Um problema do nosso cotidiano é a necessidade de estabelecer contratos sociais entre indivíduos, como por exemplo, o de prestação de serviços. O estabelecimento desse tipo de contrato envolve documentos físicos e uma série de encontros entre as partes, assim como a autenticação dos documentos por uma instituição reconhecida (cartório).
Nesse contexto, contratos estabelecidos utilizando uma rede de confiabilidade entre os usuários dispensam uma série de burocracias e aceleram a criação de tais contratos sociais. Nós já convivemos com um muitas facilidades tecnológicas bem estabelecidas, como por exemplo a abertura de contas bancárias via aplicativos de celular. Contratos inteligentes seriam mais um passo na desburocratização e simplificação de procedimentos legais e essenciais que atualmente são engessados na estrutura tradicional, e oferecem um grande espaço para auxiliar as pessoas a consolidarem seus negócios de forma rápida e barata.

Palavras-chave: Livro-razão. Blockchain. Contratos inteligentes. Transações. Segurança. Criptografia. Hash. Nonce.

Title

ABSTRACT

Blockchain technology has emerged to facilitate applications and operations that require authentication or validation.

Using a collaborative and secure system, such authentications are possible thanks to the implementation of a user network.

A problem with our daily lives is the need to establish social contracts between individuals, such as service contracts. Establishment of this type of contract involves physical documents and a series of meetings between the parties, as well as the authentication of the documents by a recognized institution.

In this context, contracts established using a trustworthiness network between users dispense a series of bureaucracies and accelerate the establishment of such social contracts.

We already live with many well-established technological facilities, such as opening bank accounts via mobile applications. Smart contracts would offer a great deal of space to help people set up their business quickly and cheaply.

Keywords: Ledger. Blockchain. Smart Contracts. Transactions. Security. Criptografy. Hash. Nonce.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Com o intuito de permitir o funcionamento do *bitcoin*, a *blockchain* nasceu em 2008 (CROSBY et al., 2015). Empreendedores e interessados da área perceberam, entretanto, que a *blockchain* poderia ser aplicada em muitas outras operações além do *bitcoin*, como, por exemplo, contratos inteligentes.

Um dos benefícios que a *blockchain* oferece é a redução dos riscos e da vulnerabilidade de transações, já que possibilita uma transação entre duas pessoas, de forma totalmente íntegra, eliminando a supervisão de uma terceira pessoa. Outros fatores muito importantes são: o poder para usuários, pois é uma aplicação transparente e descentralizada, sem nenhuma interferência de agentes externos; o fato de que todos os dados passados não são perdidos e têm1 consistência; que as informações são mais resistentes a ataques devido a criptografia e há mais privacidade, velocidade e menos custo para usuários, já que não envolve a burocaracia dos cartórios e pagamentos para terceiros. De forma geral, a *blockchain* busca simplificar transações, tornando a vida de todos mais segura, menos custosa e de fácil acesso.

Contratos inteligentes funcionam como um contrato normal, mas com todas as vantagens que uma rede *blockchain* oferece. Tais contratos não podem ser perdidos, adulterados e são auto-executáveis. Tendo isso em mente, neste trabalho foi pesquisado mais a fundo sobre contratos inteligentes, quais suas vantagens em relação aos contratos comuns feitos em cartórios e desenvolveu-se um contrato para por em prática o que foi aprendido com as pesquisas. Além disso, foi verificado a viabilidade deste tipo de contrato como um alternativa para os contratos tradicionais, avaliando a dificuldade de desenvolvimento e a complexidade de configurar o ambiente e estabelecer uma relação contratual entre duas partes utilizando contratos inteligentes via *blockchain*.

1.1 Objetivos do trabalho

Este trabalho apresenta um estudo a respeito de *blockchain* e contratos inteligentes, bem como uma breve apresentação sobre as ferramentas utilizadas e linguagem do domínio de *blockchain* escolhidas. Serão discutidos conceitos da base *blockchain* até o desenvolvimento de um contrato mínimo utilizando as assinaturas (*Address tokens*) de usuários da rede *blockchain* em um ambiente simulado demonstrando as transações como aconteceriam na prática.

Blockchain é uma tecnologia em expansão, diversos aplicativos e aplicações a estão utilizando, entretanto, muitas pessoas ainda não entendem direito como tal tecnologia funciona ou até onde ela consegue chegar, então uma das motivações para a implementação deste trabalho é entender como a tecnologia *Blockchain* funciona quando trabalhando junto com os contratos inteligentes. Além disso, contratos inteligentes não utilizam papéis, o que favorece o meio ambiente e facilita a quem usa, pois não se extraviam, estragam ou se perdem.

O objetivo deste trabalho é, portanto, implementar um contrato de prestação de serviços totalmente digital, utilizando a plataforma *Blockchain*, além de avaliar as dificuldades encontradas na implementação e utilização de tais modelos de contratos.

1.2 Organização do trabalho

Esta monografia é composta por 5 capítulos, incluindo esta introdução. No capítulo 2 são abordados os conceitos da tecnologia *blockchain*, suas classificações, seus elementos básicos, aplicações e plataformas. O capítulo 3 apresenta uma proposta para a implementação de contratos inteligentes com a tecnologia *blockchain*. A implementação e a validação desta proposta são discutidas no capítulo 4. Por fim, no capítulo 5 está a conclusão, juntamente com as dificuldades encontradas e trabalhos futuros.

2 CONCEITOS

2.1 O que é Blockchain?

A plataforma *blockchain* se baseia na ideia de redes *peer-to-peer* (P2P) e fornece um conjunto de dados compartilhado onde todos os participantes podem confiar, mesmo que não se conheçam.

Uma *blockchain* pode ser descrita como um **livro razão** (*ledger*) inviolável, compartilhado dentro de uma rede de entidades, significando que ninguém em particular a comanda e qualquer um pode inspecioná-la. Uma vez introduzidas no sistema, as informações não podem ser apagadas (CARDOSO, 2018), e todos os computadores na rede mantêm uma cópia idêntica do livro razão, que atua como um único ponto de referência. O armazenamento de dados em uma rede P2P elimina os problemas decorrentes da vulnerabilidade dos servidores centralizados, pois utiliza diferentes métodos criptográficos para proteger a rede (VOSHMGIR, 2019).

Para cada atualização, é gerada uma nova versão dos dados e uma maioria dos nós da rede P2P devem estar de acordo com esta nova versão, necessitando de um consenso entre os nós. Uma vez inseridas na *blockchain*, as informações não podem ser mais removidas. Na terminologia *blockchain*, essas informações são denominadas de transações. A tecnologia *blockchain*, portanto, depende fortemente de ferramentas de criptografia e segurança de dados, especialmente em termos de autenticação de mensagens voltados para evidência de violação e resiliência de violação.

2.2 Classificação

As arquiteturas *blockchain* são classificadas em **privada** e **pública**, também denominadas, respectivamente, de **permissionada** e **não permissionada**. As arquiteturas não permissionadas são aquelas em que a geração e a verificação de transações podem ser feitas por qualquer usuário. Elas estão presentes na maioria das moedas digitais do mercado e permitem que cada usuário crie um endereço pessoal e comece a interagir com a rede, enviando transações (DOB, 2018).

Um dos maiores exemplos de aplicação que utiliza a arquitetura não permissionada é o Bitcoin (BITCOIN, 2009), onde qualquer um é livre para baixar o aplicativo e iniciar operações de mineração e transação.

Além da possibilidade de que qualquer pessoa possa se envolver na rede *blockchain*, há outras características associadas à arquitetura não permissionada:

- necessitam ser **descentralizadas**, ou seja, nenhuma entidade possui autoridade para editar o *ledger*, desligar a rede ou alterar protocolos. Muitas redes não permissionadas, entretanto, são **baseadas em protocolos de consenso**, então alterações na rede de qualquer tipo, podem ser alcançadas desde que cinquenta por cento mais um dos usuários concordem com isso.
- a maioria das arquiteturas não permissionadas possui algum tipo de **token de incentivo ao usuário**, que pode crescer ou cair em valor, dependendo da relevância e do estado da rede *blockchain* a que pertence.
- **anonimato**, já que muitas redes não permissionadas não exigem que os usuários forneçam informações pessoais para enviar transações. Em certos casos, entretanto, informações pessoais são necessárias para fins legais, como o Bitcoin, que não oferece anonimato completo, pois a identidade do usuário está indiretamente ligada aos endereços dos quais eles têm as chaves privadas.

Como arquiteturas não permissionadas são abertas a todos, usuários mal-intencionados podem tentar publicar blocos maliciosos, ou até mesmo perigosos para o sistema e seus integrantes. Para prevenir isso, o sistema *blockchain* não permissionado utiliza um acordo multipartidário, ou consenso, requerindo que usuários gastem, ou mantenham recursos computacionais ao publicar blocos como, por exemplo, o *Proof Of Work* (BITCOIN WIKI, 2019).

Por outro lado, as arquiteturas permissionadas funcionam como ecossistemas fechados, onde é necessário a autorização para ler as informações na cadeia, os usuários não podem se conectar livremente à rede, ver o histórico gravado ou emitir transações próprias. Essa arquitetura é altamente escalonável, e é preferível por organizações centralizadas que aproveitam o poder da rede para suas próprias operações internas de negócios, garantindo também segurança nas transações.

Certas redes de *blockchain* permissionadas suportam a capacidade de revelar seletivamente informações de transações baseadas em uma identidade ou credenciais de usuários da rede. Com tal recurso é possível obter algum grau de privacidade nas transações, pois, por exemplo, pode ser que a *blockchain* registre a ocorrência de uma transação entre dois usuários, mas o conteúdo de tal transação só é acessível às partes envolvidas

Algumas das principais características das redes permissionadas incluem:

- **descentralização variável**, já que os membros da rede estão livres para negociar e chegar a uma decisão sobre o nível de descentralização que a rede terá, podendo ser tanto totalmente centralizado quanto parcialmente descentralizado.
- a possibilidade de **não haver mecanismos de consensos**, pois, para arquiteturas permissionadas, a governança é decidida pelos membros da rede de negócios.

Os modelos de consenso, por mais que não sejam extremamente necessários para redes permissionadas, quando usados, não exigem a despesa e a manutenção de recursos computacionais, como no sistema não permissionado. Isso porque é obrigatório o estabelecimento da identidade de todos os participantes de uma rede *blockchain* permissionada, mantendo um nível de confiança entre todos os integrantes, já que todos foram autorizados a publicar blocos, e tal autorização pode ser revogada se houver mau comportamento.

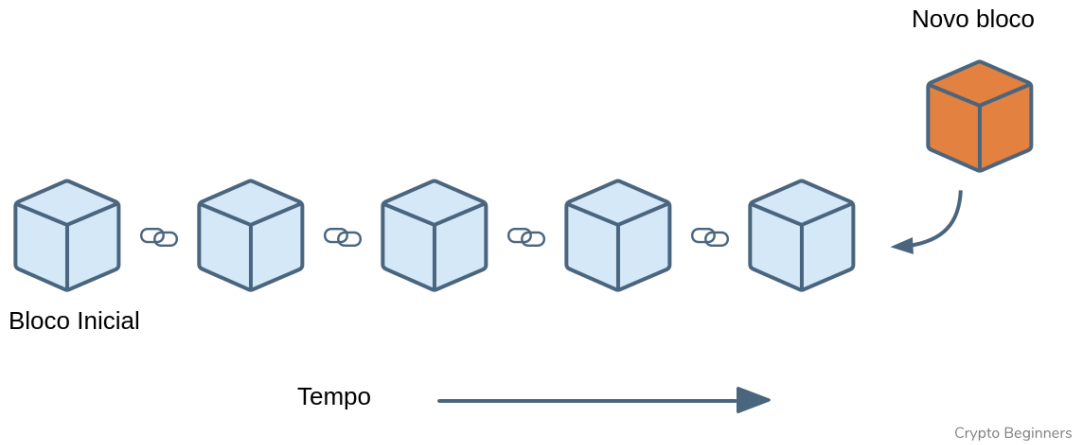
2.3 Elementos básicos da Blockchain

2.3.1 Blocos

Como pode-se ver na figura 2.1, *blockchain* é uma rede que funciona com blocos encadeados seguros que sempre carregam uma informação junto a uma impressão digital (PRADO, 2017). É um banco de dados distribuído de registros, ou razão pública, de todas as transações, ou eventos digitais, que foram realizados e compartilhados entre as partes participantes. Cada bloco, marcado com uma data, é encadeado ao bloco anterior, criando uma cadeia linear de blocos, contém 4 tipos de informações:

- A data da transação;
- Os dados, podendo ser de qualquer natureza;
- O *hash* do bloco;
- O *hash* do bloco anterior.

Figura 2.1: Relação entre tempo, bloco de início e novo bloco



Fonte: Adaptado de (CRYPTO BEGINNERS, 2019)

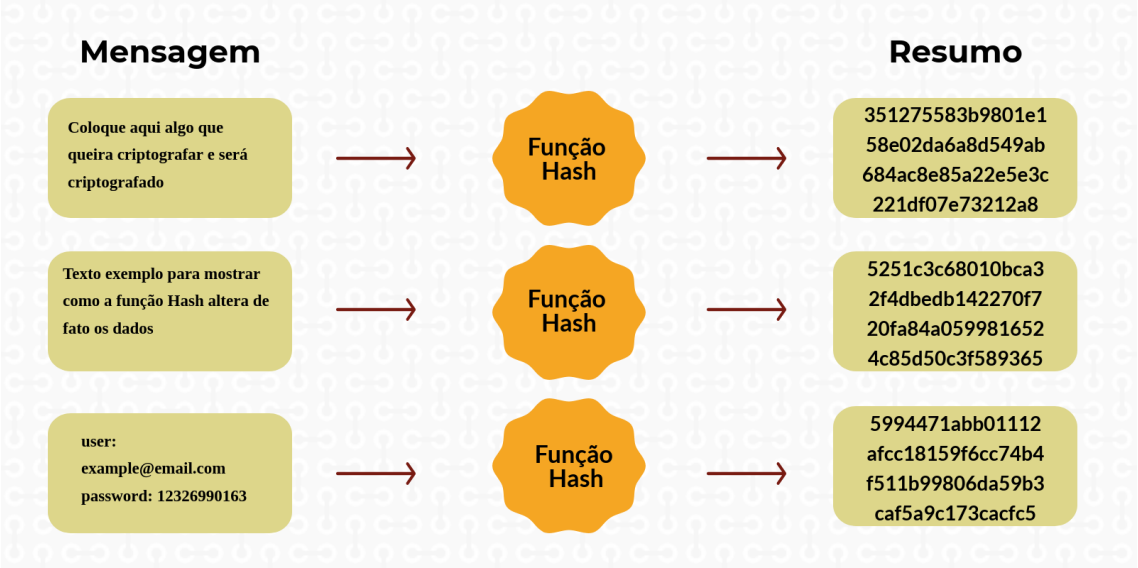
2.3.2 Funções Hash

A principal importância de uma função *hash* é garantir a **integridade** dos arquivos transmitidos, pegando um grupo de caracteres, também chamados de chave, e o mapeando para um valor de determinado comprimento.

Existem diversas funções *hash* criptograficas, *Secure Hashing Algorithm (SHA-2)*, atualmente utilizada no sistema *blockchain*, *RACE Integrity Primitives Evaluation Message Digest (RIPEMD)* e *Message Digest Algorithm 5 (MD5)*, por exemplo, e todas elas possuem quatro propriedades em comum (DANIEL, 2018):

- Computacionalmente eficientes;
- Determinísticas;
- Resistência à pré-imagem: a partir da saída de uma função *hash*, deve ser difícil encontrar o valor de entrada utilizado para gerá-la.
- Resistentes a colisões: deve ser praticamente impossível encontrar *hashes* iguais a partir de mensagens diferentes.

Figura 2.2: Exemplo de saída após a execução de uma função *hash*

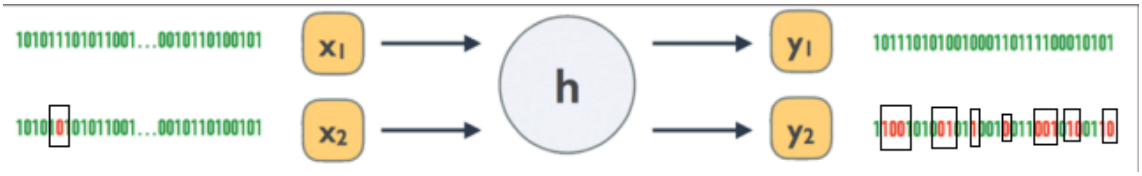


Fonte: Adaptado de (GUIA DO BITCOIN, 2017)

É possível verificar na Figura 2.2 que os valores de entrada e saída são muito distintos e, caso qualquer informação seja alterada, o *hash* é modificado e quando se gera um novo bloco contendo o *hash* do anterior, uma espécie de selo é criada, sendo possível sinalizar, e também verificar, se houve alguma alteração em um bloco e torná-lo inválido. (PRADO, 2017)

Conforme ilustrado na Figura 2.3, mesmo se apenas um valor da sequência for alterado, a saída final será muito diferente, dificultando, assim, as possibilidades de fraudes e garantindo a integridade dos dados.

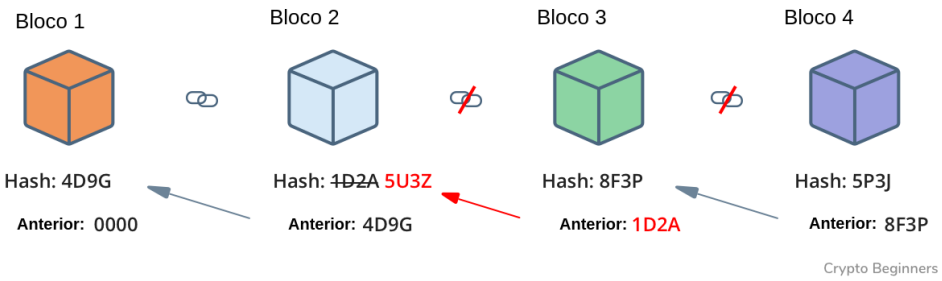
Figura 2.3: Exemplo de como uma pequena alteração na mensagem provoca uma série de mudanças na saída final



Fonte: (GANDHI; RAMASASTRI, 2017)

Foi dito anteriormente que cada bloco contém seu valor *hash* e o *hash* do bloco anterior, mas para que isso serve? Como se sabe quando houve alguma mudança? Caso haja qualquer alteração em um bloco, o seu novo *hash* será diferente daquele registrado no bloco seguinte, quebrando a corrente. Um fraudador, então, deveria recalcular o *hash* de todos os blocos que sucedem o bloco que foi alterado.

Figura 2.4: Mudanças em um *hash*



Fonte: Adaptado de (CRYPTO BEGINNERS, 2019)

Observe que na Figura 2.4, o *hash* do Bloco 2 foi alterado, resultado na quebra da corrente, já que o dado contido no bloco seguinte - o bloco 3 - não está de acordo com o anterior. Podemos observar também que, como o primeiro bloco na cadeia, conhecido como **bloco gênesis**, não possui conhecimento do bloco anterior, visto que o mesmo não existe. Nesse caso, a informação de *hash* é preenchida com zero.

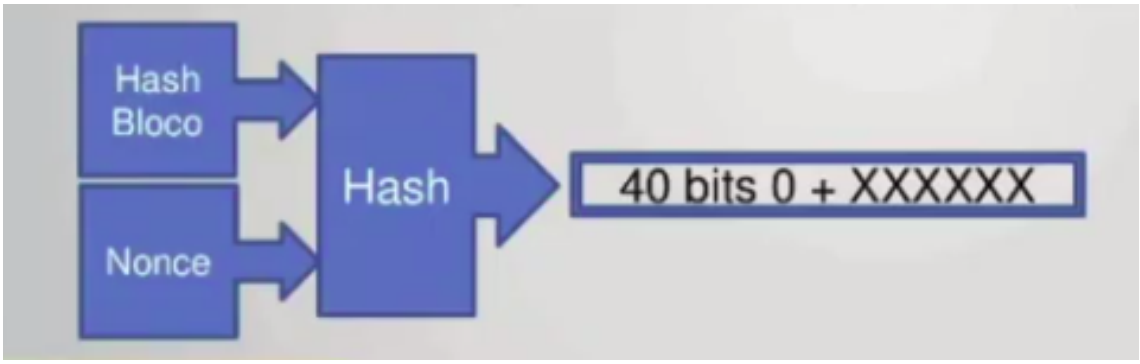
2.3.3 Consenso

A rede *blockchain* é mantida por agentes P2P, que possui uma cópia completa da versão atual da cadeia de blocos. Quando um novo bloco surge, a nova cadeia com o bloco adicional é transmitida para demais nós P2P, que possuem a função de validar a nova cadeia. Uma vez que a maioria dos pares a valide, diz-se que a rede entrou em consenso e a nova cadeia passa a ser a cadeia corrente.

Para cada tipo de arquitetura *blockchain*, existem alguns tipos de algoritmos de consenso. Em casos de *blockchain* permissionada alguns exemplos são o algoritmo Bizantino (KONSTANTOPOULOS, 2017), o RAFT (HOODA, 2019) e o *Proof of Stake* (FRANKENFIELD, 2019). Já em casos de *blockchain* não permissionada, o mais conhecido, é o *Proof of Work*, (YAGA et al., 2018) e variações do *Proof of eXercise* (LIVE COINS, 2019).

A prova de trabalho, *proof of work*, é a forma encontrada na *blockchain* para tornar as fraudes mais difíceis de serem feitas e é realizada a partir da execução de uma lógica matemática denominada de desafio. A ideia essencial é que cada bloco que será aceito no sistema *blockchain* conterá a resposta de um problema matemático específico, ou seja, o nó que gerar um bloco precisa provar que colocou recursos computacionais suficientes para resolver um problema matemático.

Figura 2.5: Exemplo de cabeçalho para o algoritmo de *Proof Of Work*



Fonte: (FAUSTINO, 2019)

Inicialmente, como mostra a Figura 2.5, cada nó monta um cabeçalho pegando o *hash* do bloco anterior e adicionando uma nova informação, o *nonce*. Um *nonce* criptográfico é um número arbitrário utilizado, apenas uma vez, que é combinado com dados para produzir o que se denomina de compilado. É possível obter diferentes valores de compilado apenas alterando o valor *nonce*, mas mantendo os mesmos dados.

O desafio proposto é encontrar o valor *hash*, onde os primeiros bits dessa informação se iniciam em zero e o final é concatenado ao próprio *hash* do bloco. Tal procedimento é feito via força bruta, variando o valor *nonce*. Quanto maior o número de bits necessários para encontrar o *hash* com os valores zero no início, maior a dificuldade de resolver o desafio. A quantidade de bits varia a cada novo bloco, em função da rapidez com que o desafio está sendo resolvido. O Bitcoin, por exemplo, tenta sempre manter a criação de novos blocos em uma média de tempo de 10 minutos. Se o algoritmo percebe que o desafio está sendo resolvido pela rede em um tempo menor, ele aumenta a quantidade de zeros no início, dificultando o problema e fazendo com que o tempo de resolução tenda a 10 minutos. Se a rede percebe que o desafio está demorando mais do que a média, é reduzido o número de zeros.

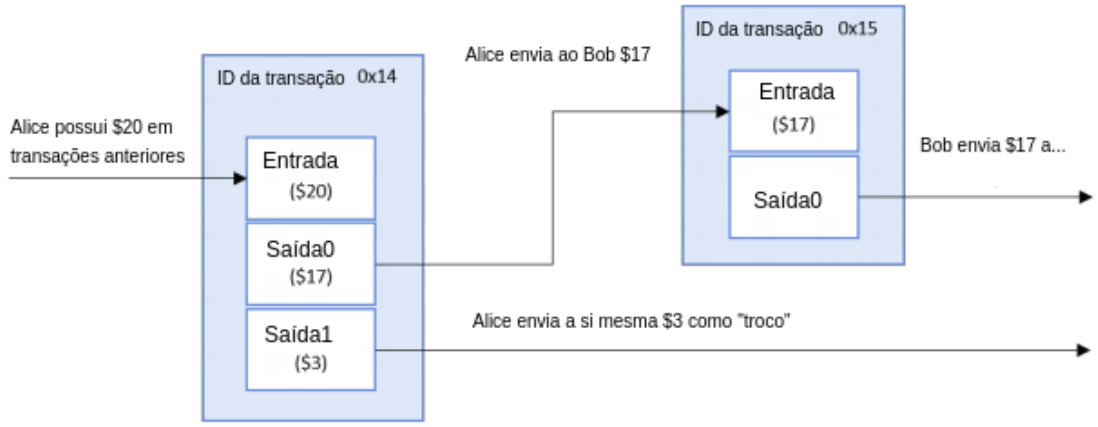
Os nós que doam seus recursos computacionais para resolver os enigmas e gerar blocos são chamados de **mineiros** (*miners*) e são financeiramente premiados pelos seus esforços. Tal processo de doar recursos é conhecido como **mineração**.

2.3.4 Transações

No sistema *blockchain*, um contrato mútuo entre qualquer conjunto de entidades é, geralmente, denominado como transação. Como é visto na Figura 2.6, as transações

com criptomoedas representam uma transferência da criptomoeda entre os usuários da rede *blockchain*, já para um cenário de negócios, uma transação pode ser uma maneira de registrar as atividades ocorridas de forma digital ou física.

Figura 2.6: Exemplo de transação com criptomoedas



Fonte: Adaptado de (YAGA et al., 2018)

Os dados que compreendem uma transação podem ser diferentes para cada implementação *blockchain*, mas o mecanismo para transacionar é basicamente o mesmo. As informações enviadas por um usuários para a rede *blockchain* podem incluir o **endereço do remetente** (ou outro identificador relevante) e sua **chave pública**, além de **assinatura digital** e **entradas e saídas de transações**.

Independente de como os dados são formados e transicionados, é muito importante determinar a validade e autenticidade de uma transação. A **validade** da transação garante que ela atenda aos requisitos de protocolos e quaisquer formatos de dados formalizados, ou requisitos do contrato inteligente, para implementação da *blockchain*. Já a **autenticidade** de uma transação determina que o remetente dos dados digitais é de fato dono de tais dados.

A rede *blockchain* utiliza **criptografia de chaves assimétricas**, (YAGA et al., 2018), ou seja, as transações são tipicamente assinadas digitalmente pela chave privada do remetente e podem ser verificadas a qualquer instante utilizando sua chave pública. Pode-se verificar, portanto, que um sistema *blockchain* é um *ledger* inviolável e distribuído de transações.

Tais procedimentos com criptografia assimétrica permitem uma relação de confiança entre usuários que não se conhecem, já que a criptografia de transação com a chave privada, por exemplo, prova que o assinante de uma transação possui acesso à chave privada.

Em algumas redes *blockchain*, especialmente as não permissionadas, os usuários devem gerenciar e armazenar com segurança suas próprias chaves privadas. Para armazená-las com segurança, eles utilizam um software conhecido como carteira (*wallet*) que pode armazenar chaves públicas, privadas e endereços associados.

Como é computacionalmente inviável regenerar uma mesma chave privada, se algum usuário perder a sua, qualquer documento digital associado a essa chave será perdido, já que não há garantias de que a chave não foi roubada e, caso tenha sido esse o ocorrido, o invasor terá acesso total a todos os documentos controlados pela chave privada.

O armazenamento de chaves privadas é um aspecto muito importante da tecnologia *blockchain*, pois, por exemplo, o roubo dessas pode implicar na realização de transações que enviem dinheiro para alguma conta externa, causando prejuízos e comprometendo a verdadeira conta em questão. Como as operações na *blockchain* não podem ser desfeitas, uma vez que o criminoso consiga chaves privadas e transfira publicamente fundos associados a outra conta, não será possível reverter tal transação.

2.4 Aplicações de Blockchain

2.4.1 Criptomoedas

As criptomoedas permitem um novo sistema de gerenciamento monetário robusto, transparente e eficiente.

Do ponto de vista da tecnologia, os sistemas monetários existentes exigem dinheiro em papel ou utilizam um serviço privado de terceiros (cartões de crédito e débito) para suportar transações globais. Do ponto de vista econômico, os detentores de moedas emitidas pelo governo devem confiar nas autoridades centralizadas que as avaliações monetárias gerais permancerão estáveis e que transferências ou participações online não podem ser apreendidas. Não há necessidade de uma parte centralizada controlar uma criptomoeda, nem existe qualquer tipo de restrições ou regras de uso. (BLOCKCHAIN-TECHNOLOGIES.COM, 2019b).

2.4.2 Contratos inteligentes

Criados em 1994 por Nick Szabo (ROSIC, 2016b), os contratos inteligentes, *smart contracts*, são uma boa ideia para a execução automática de contratos entre partes participantes, já que, depois de escrito, é um documento que não pode ser alterado e autoriza, ou não, transações de acordo com os termos estabelecidos.

Contratos inteligentes são um conjunto de códigos de computador, entre duas, ou mais, partes executadas no topo de uma rede *blockchain* e constituem um conjunto de regras acordadas pelas partes envolvidas. Na execução, se esse conjunto de regras predefinidas for atendido, o contrato inteligente será executado para produzir a saída. Esse trecho de código permite a automação descentralizada, facilitando, verificando e aplicando as condições de um contrato subjacente. Os contratos inteligentes permitem a troca de qualquer coisa de valor, como por exemplo, dinheiro, ações ou propriedades, de maneira transparente, eliminando a necessidade de um intermediário e mantendo o sistema livre de conflitos (PRATAP, 2018).

Eles são **imutáveis**, uma vez que estão dentro das cadeias de blocos e **distribuídos**, já que estão dentro da *blockchain*. Abaixo seguem alguns exemplos de aplicações para os contratos inteligentes.

Eleições: os contratos inteligentes forneceriam um sistema de votação infinitamente mais seguro, evitando manipulações. Os votos protegidos pelo *ledger* precisariam ser decodificados e exigiriam um excessivo poder computacional para acessá-los, já que, assim que os votos fossem inseridos na *blockchain*, todos os nós da rede seriam atualizados para refletir as novas informações.

Aplicativos como o **FollowMyVote** (FOLLOWMYVOTE, 2019) usam contratos inteligentes e tecnologia *blockchain* para proteger votos contra fraudes. Quando a transação de votação é gravada na *blockchain*, ela não pode ser alterada, assim, quando a votação terminar, o contrato inteligente enviará um *token* para um endereço que represente o vencedor da votação (BITDEGREE, 2019).

Logística e cadeia de suprimentos: Além das cadeias de suprimento tradicionais serem alvos de burocracias, elas necessitam passar por canais de aprovação, que aumentam a possibilidade de perdas e fraudes, elas também são improdutivas e incluem muitas ligações. De forma a automatizar as tarefas e pagamentos, a execução de contratos inteligentes na *blockchain* anularia os contras citados acima, fornecendo uma versão digital, segura e acessível a todos os envolvidos na cadeia (CARDOSO, 2018).

A empresa ***Barclays Corporate Bank*** (PRISCO, 2015) utiliza contratos inteligentes para registrar uma mudança de propriedade e transferir automaticamente pagamentos para outras instituições financeiras na chegada (ROSIC, 2016b).

Cuidados de saúde: Registros pessoais de saúde poderiam ser codificados e armazenados na *blockchain* com uma chave privada que concederia acesso apenas a indivíduos específicos. Recibos de cirurgia podem ser armazenados em uma rede *blockchain* e enviados automaticamente para provedores de seguro de saúde, como prova de entrega. Além disso, o *ledger* poderia ser usado para gerenciamento geral de saúde, como supervisão de medicamentos, conformidades com regulamentos, resultados de testes e gerenciamento de suprimentos de saúde (ROSIC, 2016b).

Já existem exemplos de contratos inteligentes sendo utilizados no setor médico por empresas como a ***Encrypgen*** (ENCRYPGEN, 2018), que utiliza contratos inteligentes para transferir dados de pacientes de forma segura, não permitindo o acesso de terceiros. Dessa forma, os pacientes estão no controle de seus próprios dados: se os pesquisadores querem usar tais dados, eles devem pagar por eles e quem escolhe se os vende ou não é o próprio paciente (BITDEGREE, 2019).

Apólices de seguro: Com contratos inteligentes, as companhias de seguro podem automatizar suas políticas, já que, caso as condições de entrada do contrato mudem em um evento segurado, como uma catástrofe natural, por exemplo, o processo de reivindicações é imediatamente desencadeado. Parâmetros mensuráveis de tal evento, como a velocidade do vento, ou a magnitude de um terremoto, podem ser registrados na cadeia de blocos e, à medida que os parâmetros passem de certos limiares pré-acordados, o processo de reclamações é desencadeado imediatamente e, sem necessidade de intervenções humanas, a quantidade exata de pagamento financeiro pode ser entregue (ROSIC, 2016b).

Em 2017, na França, duas companhias de seguros, a ***Atlas Insurance Malta*** e a ***Axa*** testaram contratos inteligentes. Eles possuíam protótipos que compensavam os clientes das companhias aéreas se seus vôos atrasassem (BITDEGREE, 2019).

Conteúdo protegido por direitos autorais: No mundo da música, o artista e a editora possuem direitos sobre o conteúdo. Sempre que o conteúdo é utilizado para fins comerciais, o titular deve receber uma taxa de *royalties*. O problema com o atual sistema é saber quem possui os direitos e garantir que todos aqueles que estão legalmente permitidos a receber o pagamento dos *royalties* os recebam efetivamente. A utilização de contrato inteligente, como ilustrado na Figura 2.7, manteria o controle de todos os direitos de propriedade, garantindo confiança na propriedade verdadeira, já que qualquer alteração nos

dados exige um consenso de todas as partes na rede. Conhecendo sempre o verdadeiro detentor da propriedade, o contrato inteligente garantiria que os *royalties* fossem gerados e pagos em tempo real (ROSIC, 2016b).

Figura 2.7: Exemplo de contrato inteligente: controle dos direitos de propriedade

Fonte: (CARDOSO, 2018)

2.4.3 Propriedades inteligentes

Carros, casas ou até mesmo títulos de propriedade ou ações de empresa podem conter tecnologia inteligente. Tal registro pode ser armazenado no livro razão juntamente com detalhes contratuais de outras pessoas que têm permissão de uso nesta propriedade. Chaves inteligentes podem ser utilizadas para facilitar o acesso à parte permitida e o livro razão armazena e permite a troca dessas chaves inteligentes após a validação do contrato.

O livro razão descentralizado também se torna um sistema para registrar e gerenciar direitos de propriedade, além de permitir que os contratos inteligentes sejam duplicados se os registros ou a chave inteligente forem perdidos. Tornar a propriedade inteligente diminui os riscos de fraudes e situações comerciais questionáveis, além de aumentar a confiança e a eficiência. (ROSIC, 2016a)

2.4.4 Internet das coisas (*IoT*)

A tecnologia *blockchain* pode ser utilizada no rastreamento de bilhões de dispositivos conectados, permitindo o processamento de transações e a coordenação entre dispositivos. Tal abordagem descentralizada eliminaria pontos únicos de falha, criando um ecossistema mais resiliente para os dispositivos funcionarem, além de tornar os dados dos consumidores mais privados graças aos algoritmos criptográficos (BLOCKCHAIN-TECHNOLOGIES.COM, 2019a). Abaixo seguem alguns exemplos de aplicações.

Aparelhos inteligentes: inúmeras residências e automóveis agora vêm com aparelhos que podem se conectar a outros aparelhos, aplicativos da Internet e celulares. Em vez de armazenar esses dados em um servidor central ou em uma solução de armazenamento baseada na nuvem, tais dados podem ser armazenados na *blockchain*, protegendo então as informações pessoais.

Cadeia de suprimentos: devido à falta de transparência e complicações na cadeia de suprimentos e logística atuais, a *blockchain* e a *IoT* combinadas podem ajudar a melhorar a confiabilidade e a rastreabilidade da rede. Os sensores de *IoT*, como movimento, GPS, temperatura, informações do veículo ou dispositivos conectados, fornecem detalhes nítidos sobre o status das remessas. As informações do sensor são então armazenadas na *blockchain*. Depois que os dados são salvos na *blockchain*, as partes interessadas listadas nos contratos inteligentes obtêm acesso às informações em *online* (TAKYAR, 2019).

2.5 Plataformas Blockchain

Há várias plataformas para o desenvolvimento de aplicação baseada em *blockchain*. Para as arquiteturas **permissionadas** existem as seguintes.

Hyperledger Fabric (HYPERLEDGER, 2018): Fundada em 2016, é uma base para o desenvolvimento de aplicativos, ou soluções com uma arquitetura modular, de código aberto e projetada para uso em contextos empresariais.

Existem dois diferenciais sobre outras plataformas: o primeiro é que ele foi estabelecido sob a **Linux Foundation** (THE-LINUX-FOUNDATION, 2019), que possui uma longa história de fomentar projetos de código aberto sob governança aberta, e o segundo, é seu suporte a protocolos de consensos conectáveis, que permitem que a plataforma seja customizada de maneira mais eficaz para atender a casos de uso e modelos de confiança específicos.

O **Hyperledger Fabric** possui uma arquitetura altamente modular e configurável, permitindo inovação, versatilidade e otimização para uma ampla gama de casos de uso da indústria. Além disso, é a primeira plataforma *blockchain* a oferecer suporte a contratos inteligentes criados em linguagens de programação de uso geral, como **Java, Go e Node.js**, facilitando então o desenvolvimento de contratos inteligentes por empresas, já que não é necessário treinamento adicional para aprender uma nova linguagem.

Corda (CORDA, 2019): Criado pelo R3 (R3, 2019) em 2017, é uma plataforma de código aberto que permite que as empresas realizem transações diretamente, e em estrita privacidade, usando contratos inteligentes, reduzindo os custos de transação e manutenção de registros e simplificando as operações comerciais. O Corda utiliza um protocolo ***need-to-know basis***, ou seja, não fala todas as transações para todos os membros da rede, mas apenas as envia para quem faz parte de cada transação. Outra grande diferença é que os contratos inteligentes podem ter seu código atualizado.

Chain Core (CHAIN..., 2016): É uma plataforma que ajuda a iniciar e transferir ativos financeiros com base na permissão da infraestrutura *blockchain*. Suporta múltiplas linguagens de programção, tais como **Java, Node.js e Ruby** e oferece suporte à geração e armazenamento de chaves privadas e assinaturas digitais em transações autorizadas.

Já para arquiteturas *blockchain* **não permissionadas** as plataformas mais conhecidas são o **Ethereum** e a **Bitcoin**. Essas plataformas são apresentadas a seguir.

Ethereum (ETHEREUM, 2019): Criada em 2015 por Vitalik Buterin, é uma plataforma global, de código aberto e para aplicativos descentralizados. Como outras aplicações *blockchains*, o Ethereum possui uma criptomoeda nativa chamada **Ether** (ETH), um tipo de *token* criptográfico que alimenta a rede.

Não existe uma empresa ou organização centralizada que controle o Ethereum. Ele é mantido e aprimorado ao longo do tempo por uma comunidade global que trabalha desde o protocolo principal até os aplicativos do consumidor.

O **Ethereum** é programável, o que significa que os desenvolvedores podem usá-lo para criar novos tipos de aplicativos, que ganham os benefícios da tecnologia de criptomoeda e *blockchain*. Tais aplicativos podem ser confiáveis, ou seja, assim que forem "enviados"para o **Ethereum**, eles sempre serão executados conforme programado, podem controlar ativos digitais para criar novos tipos de aplicativos financeiros e podem ser descentralizados, significando que nenhuma entidade ou pessoa os controla (ROSIC, 2016c).

Bitcoin (BITCOIN, 2009): Criada em 2009, a plataforma utiliza tecnologia ponto a ponto para operar sem autoridade central ou bancos: o gerenciamento de transações e a emissão de *bitcoins* são realizadas coletivamente pela rede. A **Bitcoin** é de código aberto e seu design é público, ninguém é dono ou controla a **Bitcoin** e todos podem participar.

A **Bitcoin** permite que o desenvolvedor crie novos serviços *online* que antes não existiam devido a limitações financeiras, tais como sistemas de gorjetas e soluções de pagamento automatizadas. Além disso, a **Bitcoin** cria um endereço exclusivo para cada transação, ou seja, para criar um sistema de pagamento associado a uma fatura, tudo o que precisa ser feito é gerar e monitorar um endereço de *bitcoin* para cada pagamento, sem nunca utilizar o mesmo endereço para mais de uma transação.

2.6 Resumo

Neste capítulo foi visto que a *blockchain* é um **livro razão** inviolável e compartilhado dentro da rede, que funciona com blocos encadeados que carregam informações junto de sua impressão digital.

Para que o sistema *blockchain* funcione, é necessário ferramentas de criptografia e de segurança dos dados, como as **funções *hash***, que transformam um valor inicial em um resultado muito diferente. Tais funções dificultam as possibilidades de fraudes e garantem a integridade dos dados, já que caso qualquer bloco tenha seu conteúdo modificado, o seu valor *hash* será diferente do registrado anteriormente e o fraudador terá então que recalcular o *hash* de todos os blocos que sucedem o bloco que foi alterado.

O sistema *blockchain* utiliza um protocolo de **consenso** que permite que a *blockchain* seja atualizada, garantindo que todos os blocos da cadeia são verdadeiros e também evita que uma única entidade controle o sistema *blockchain*. O principal requisito para obter um consenso é a aceitação de cinquenta por cento mais um entre os nós da rede. O algoritmo que foi visto neste trabalho foi a **prova de trabalho**.

A arquitetura *blockchain* pode ser **permissionada** ou **não permissionada**. Arquiteturas permissionadas obrigam autorização para ler as informações na cadeia ou emitir transações e podem ser governadas pelos próprios membros da rede de negócios, não necessitando de um algoritmo de consenso. Tais arquiteturas são mais comuns entre empresas e negócios, onde segurança, identidade e definições de funções são importantes.

Já arquiteturas não permissionadas permitem que usuários, assim que criarem um endereço pessoal, possam interagir com a rede, executar um nó no sistema ou empre-

gar os protocolos de mineração para ajudar na verificação de transações. O maior exemplo de tecnologia *blockchain* que utiliza esta arquitetura é o Bitcoin.

Foi visto também que para as **transações** serem executadas, é necessário garantir a validade da transação e sua autenticidade. Tais garantias são feitas utilizando **criptografia de chaves assimétricas** e as mesmas são armazenadas em um software chamado de carteira, dificultando o roubo e inviolações.

Por fim foram vistas algumas aplicações da rede *blockchain*, como criptomoedas e os **contratos inteligentes**, que podem facilitar diversas áreas, como as de seguro, saúde e até em processos eleitorais, e alguns de seus usos.

3 PROPOSTA

Sabendo que contratos físicos são burocráticos, custosos, consomem tempo e necessitam de diversas validações manuais, neste trabalho propõe a implementação de um **contrato virtual**, com documentação eletrônica, para prestação de serviços e verificar suas dificuldades de implementação e uso, analisando se contratos virtuais são de fato melhores do que os contratos físicos.

Na teoria, contratos virtuais possuem inúmeras vantagens em relação aos contratos físicos normais (GRYBNIAK, 2017), tais como **confiabilidade**, pois os documentos são criptografados em um *ledger* compartilhado, havendo garantias de certeza, segurança, transparência e legitimidade dos processos automatizados, além de não ser passível de perda; **segurança**, já que não há como um contrato inteligente ser perdido ou alterado sem permissão, pois é criptografado e distribuído pelos nós da rede; **velocidade**, porque contratos inteligentes automatizam tarefas utilizando códigos de *software*, o que reduz horas de processos negociais; e **preço**, pois como não há o envolvimento de terceiros cobrando taxas, os contratos inteligentes reduzem custos para o consumidor.

Com exceção da negociação de preços e prazo para prestação do serviço, nenhuma atividade necessita ser presencial. Por questões de segurança, contratos virtuais não podem ser alterados assim que criados, então caso haja interesse de alguma das partes em modificar o preço ou o prazo máximo de entrega, um novo contrato virtual deverá ser construído.

3.1 Modelo de contratos físicos

Pode-se verificar na Figura 3.1 que em um contrato de prestação de serviço físico, tanto contratante quanto contratado precisam ser identificados através de diversas informações autenticadas. Em um contrato virtual tais identificações são feitas apenas com a chave de cada um dos participantes.

Figura 3.1: Modelo de contrato de prestação de serviços: da parte de identificação

CONTRATO DE PRESTAÇÃO DE SERVIÇOS E HONORÁRIOS DE PROFISSIONAL AUTÔNOMO

Contratante: (Nome), (nacionalidade), (estado civil), (profissão), portador da cédula de identidade R.G. nº xxxxxx e inscrito no CPF/MF nº xxxxxxx, residente e domiciliado na (Rua), (número), (bairro), (CEP), (Cidade), (Estado);

Contratado : Nome), (nacionalidade), (estado civil), (profissão), portador da cédula de identidade R.G. nº xxxxxx e inscrito no CPF/MF nº xxxxxxx, residente e domiciliado na (Rua), (número), (bairro), (CEP), (Cidade), (Estado);

Fonte: (LEX MAGISTER, 2019)

Referente às tarefas propostas, o contrato virtual implementado se comporta de forma similar a um contrato físico, como podemos ver na Figura 3.2, com a diferença de que as funções a serem realizadas serão citadas no **código**, não sendo passíveis de mudanças após a criação do mesmo.

Figura 3.2: Modelo de contrato de prestação de serviços: da parte de serviços

DOS SERVIÇOS

CLÁUSULA 3ª : O contratado prestará os seguintes serviços (descrever detalhadamente os serviços prestados).

Fonte: (LEX MAGISTER, 2019)

Para fins de teste, o contrato implementado possui um tempo limite para a realização do mesmo, diferente de um contrato físico cujo prazo é indeterminado, mas com antecedência mínima de trinta dias para o caso de rescisão, como pode-se ver na Figura 3.3.

Figura 3.3: Modelo de contrato de prestação de serviços: da parte de honorários

CLÁUSULA 9ª: O presente contrato, terá vigência por prazo indeterminado, porém, havendo interesse em sua rescisão, a parte interessada notificará a parte contraria, por escrito, com antecedência mínima de trinta (30) dias.

Fonte: (LEX MAGISTER, 2019)

Para avaliar as vantagens e desvantagens de implementar um contrato virtual na prática, será feita uma análise sobre as dificuldades encontradas ao longo do trabalho, os conhecimentos necessários para entender como funcionam, além de testes de casos de uso com diferentes cenários.

3.2 Requisitos funcionais e não funcionais

Dentro da engenharia de software, requisitos funcionais são aqueles que expecificam funções que o sistema deve ser capaz de realizar (FILHO, 2008), sem se preocupar com o 'como' isso vai acontecer, mas sim definir a expectativa do que o *software* irá fazer para satisfazer aquilo que os usuários desejam, ou seja, as necessidades do negócio (MARINHO, 2015).

Já os requisitos não funcionais dizem respeito às características e padrões de qualidade que o sistema deve oferecer (MARINHO, 2015) e não estão diretamente relacionados à funcionalidade do sistema (FILHO, 2008). A seguir estão detalhados os requisitos funcionais e não funcionais para o contrato virtual implementado.

Requisitos funcionais:

Possibilidade de criação do contrato: aquele que deseja abrir o contrato deve ser capaz de criá-lo em qualquer plataforma que deseje, desde que utilizando o código binário do contrato e do formato JSON (GAMA, 2011) resultante do mesmo. Tais informações estão disponíveis no contrato e não necessitam de conhecimentos técnicos avançados do contratante;

Visualizar constantes como preço do serviço, prazo limite para entrega, estado atual do contrato e prazo restante de entrega: contratante e contratado devem possuir acesso para verificar a qualquer momento qual o valor proposto para a prestação de serviço, quanto tempo o contratado possui para sua realização, quanto tempo o contratado ainda possui para a realização do serviço e em que estado o contrato está;

Informar que realizou o serviço: o contratado deve ser capaz de informar que o serviço foi realizado;

Informar que recebeu o serviço: o contratante deve ser capaz de informar que o serviço foi realizado;

Receber o depósito: o contratado deve receber o valor acertado na criação do contrato assim que o contratante informar que o serviço foi de fato prestado;

Estorno das garantias: na criação do contrato, tanto contratante quanto contratado pagam uma garantia para que o contrato seja de fato dado como contratado. Caso uma das partes não pague, é possível que aquela quem pagou pegue o que pagou de volta e cancele o contrato.

Requisitos não funcionais:

Disponibilidade: ambas as partes podem, a qualquer momento, acessar e visualizar o *status* atual do contrato, desde que cumpram o primeiro requisito de conexão com *Internet*, visto que ele está disponível na rede *blockchain*.

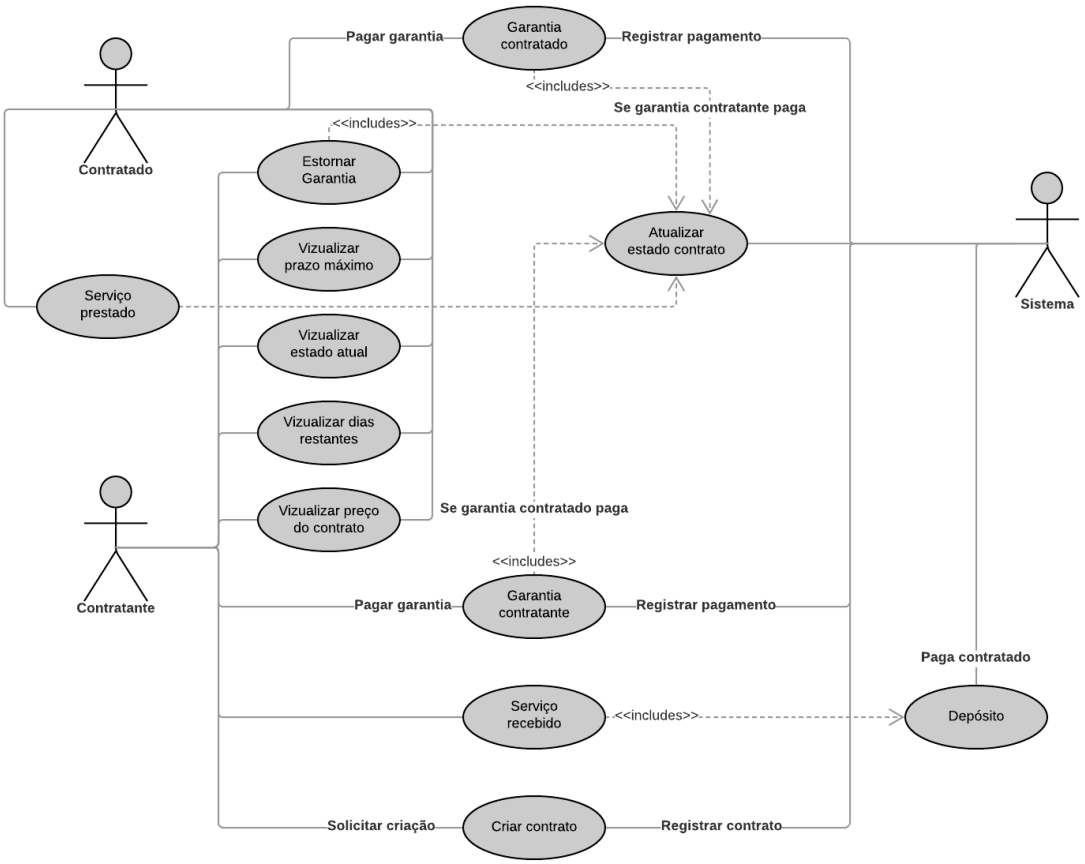
A disponibilidade é um fator importante para o escopo de contratos virtuais, já que ele deve ser acessível a todos os indivíduos envolvidos. A rede *blockchain* garante essa disponibilidade através de seu primeiro pilar, **descentralização**, já que ela é *peer to peer*, desvinculando a necessidade de um servidor dedicado estar ligado constantemente prestando serviço. Além do pilar de descentralização , existem ainda outros dois pilares que são fundamentais para a rede *blockchain* e muito convenientes para a criação de contratos virtuais: a **transparência** e **imutabilidade** (o código não é passivel de mudanças, reduzindo fraudes).

3.3 Diagrama de Casos de Uso

Diagramas de Casos de Uso documentam o que o sistema faz do ponto de vista do usuário, ou seja, descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com seus usuários. Em tais diagramas não há o aprofundamento sobre detalhes técnicos e são compostos basicamente por quatro partes: o **cenário**, a sequência de eventos; o **ator**, a parte interessada no sistema; o **caso de uso**, a tarefa realizada; e a **comunicação**, ligação entre o ator e o caso de uso (RIBEIRO, 2012).

O trabalho foi fundamentado para efetivação de um contrato para prestação de serviços entre contratante e contratado, com prazos de entrega, possibilidade de estorno das garantias e transferências monetárias. O contrato tem foco em tarefas domésticas e as funções a serem cumpridas estão inseridas dentro dele, não sendo possíveis alterações, tornando o contrato único para tal serviço. A Figura 3.4 apresenta o diagrama de Casos de Uso elaborado sobre os requisitos do contrato.

Figura 3.4: Diagrama de Casos de Uso



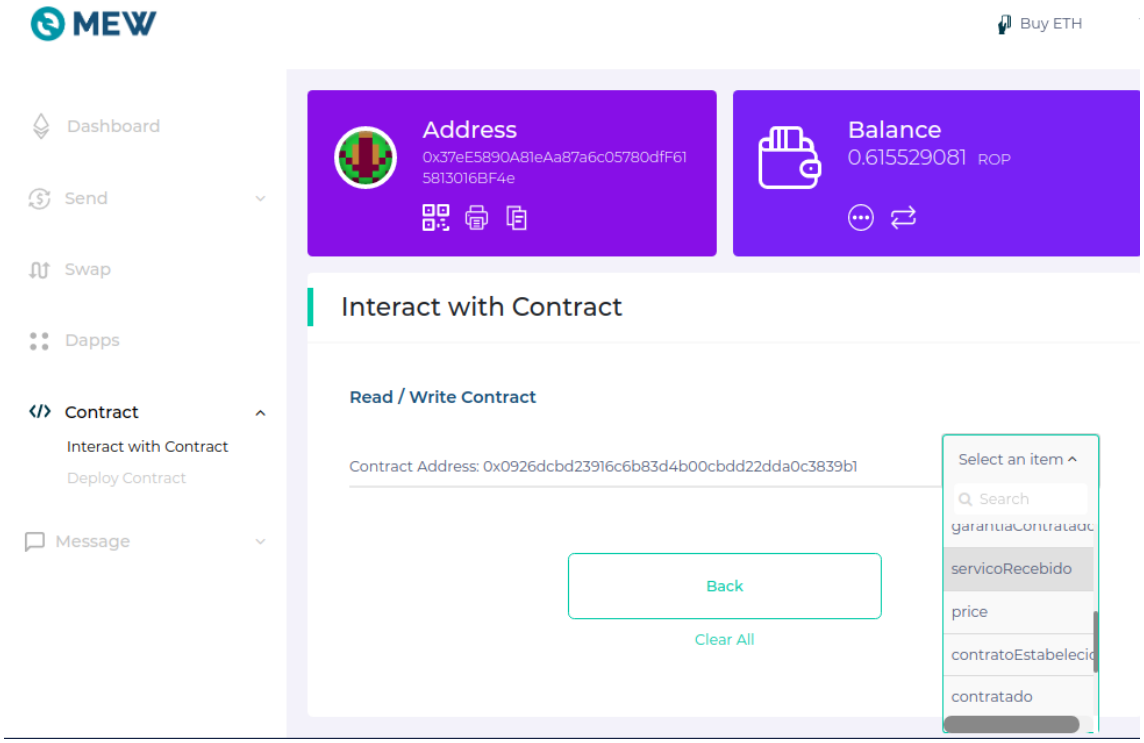
Fonte: Autor

3.4 Interface do Sistema

Não foi implementada nenhuma interface para a utilização do contrato virtual, pois já existem ferramentas que permitem uma boa visualização.

A Figura 3.5 ilustra a plataforma utilizada, na hora de interagir com o contrato. O quadrado superior esquerdo mostra a identificação do usuário, já o da direita mostra o total de "dinheiro"que o usuário possui em sua carteira virtual. O valor que vem após *Contract Address* é o identificador do contrato e as possibilidades de ações estão na caixinha a direita. No exemplo mostrado, o usuário está prestes a selecionar a ação de informar que o serviço foi recebido.

Figura 3.5: Interface para interação com o contrato



Fonte: Autor

3.5 Considerações Finais

Neste capítulo apresentou-se uma introdução aos contratos virtuais e uma comparação com contratos físicos, além dos requisitos funcionais e não funcionais para o contrato implementado, seus casos de uso e a interface utilizada.

4 IMPLEMENTAÇÃO E AVALIAÇÃO

Este capítulo tem como objetivo discutir a implementação da proposta e apresentar os resultados obtidos após alguns testes de validação. Para ter uma base de por onde começar, tal implementação foi baseada no tutorial visto na página **Agatetepê** (AGATETEPê, 2018).

Devido a necessidade de envolver dinheiro real, o contrato virtual implementado será colocado apenas em uma rede *blockchain* de teste, a ***Ropsten Test Network*** (STACKEXCHANGE, 2017).

4.1 Soluções tecnológicas

Para a implementação desde trabalho, utilizou-se **Ethereum** como plataforma, a linguagem **Solidity**, **Remix** como ambiente de desenvolvimento e **My ether wallet** para a manipulação do contrato.

Ethereum (ETHEREUM, 2019): Como já dito anteriormente, é uma plataforma descentralizada que implementa contratos inteligentes. Tais contratos são executados em uma máquina virtual (*Ethereum Virtual Machine*), uma rede de computação distribuída composta por todos os dispositivos que executam os nós da Ethereum, motivo pela qual a plataforma foi escolhida para a implementação do trabalho. Para executar um contrato inteligente nos nós da *blockchain* o interessado deve pagar aos operadores de tais nós em ***Ether***, a criptomoeda associada ao **Ethereum**. Para tanto foi utilizado um sistema de testes que será visto mais adiante, na seção 4.2.

Solidity (SOLIDITY, 2016): É uma linguagem de alto nível, baseada em C++, Python e JavaScript, orientada a objetos e criada para desenvolver contratos inteligentes. É estaticamente tipada, suporta heranças, bibliotecas e tipos complexos definidos pelo usuário, entre outros recursos. Foi escolhida para a implementação do trabalho devido ao vasto número de exemplos que existem na Internet e por ser parecida com JavaScript, linguagem atualmente utilizada pela autora.

Remix (REMIX-IDE, 2019): É um ambiente de desenvolvimento que permite escrever, compilar e depurar códigos escritos em Solidity. É uma plataforma escrita em JavaScript, sem nenhum *framework* adicional, que suporta tanto o uso no navegador quanto localmente e foi criada pela comunidade *blockchain* e pode ser encontrada em (REMIX, 2019).

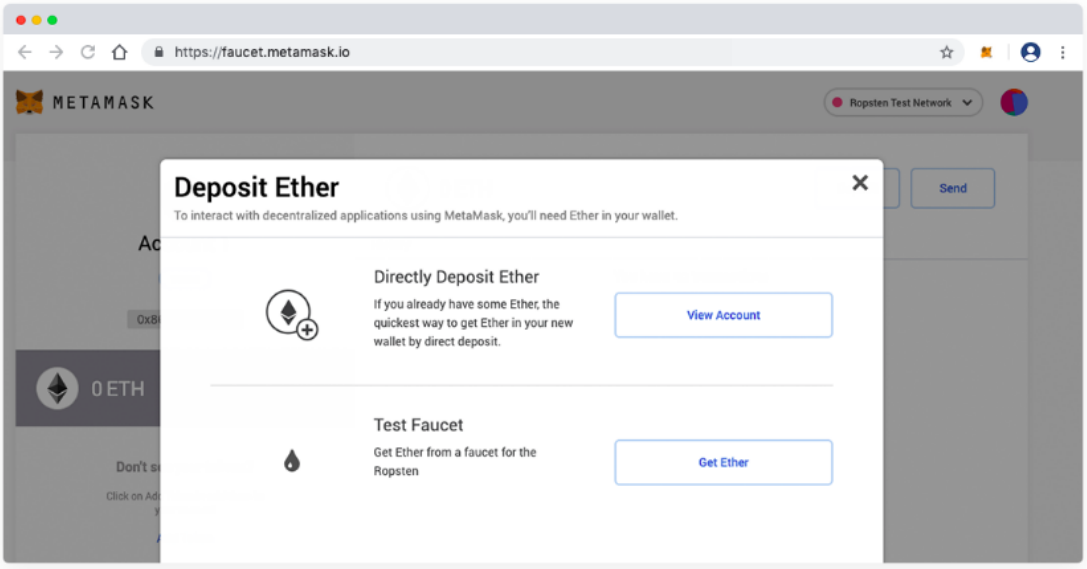
MetaMask (METAMASK, 2019): É um aplicativo que permite a execução de aplicativos Ethereum. Ele inclui um cofre de identidade seguro, fornecendo uma interface de usuário para gerenciar suas identidades em diferentes sites e assinar transações de *blockchain*. Foi escolhido neste trabalho por ter uma extensão disponível no navegador **google chrome** e por ser simples de utilizar.

My Ether Wallet (MYETHERWALLET, 2019): É uma carteira de criptomoedas *online* onde o usuário tem total controle de suas senhas, fundos e chaves. Foi escolhida para manipular o contrato principalmente devido a flexibilidade e facilidade de manuseio, entretanto, quando se fala de segurança, ***My Ether Wallet*** não seria a primeira opção, já que o usuário é totalmente responsável por seus dados e a possibilidade de perdas é alta. Como o contrato implementado é inicialmente, apenas uma prova de conceitos, tais questões de segurança, apesar de importantes, não são prioritárias.

4.2 Configuração do ambiente

Como foi visto na seção 4.1, o **Ethereum** necessita o pagamento de *ethers* para qualquer operação que seja feita, então foi necessário conseguir alguns. O aplicativo **MetaMask** permite que o usuário escolha se quer permanecer na rede oficial da *blockchain*, que custa dinheiro, ou em algumas de testes que ele disponibiliza. Foi escolhida a rede de teste **Ropsten Test Network**, que permite conseguir *ethers* falsos.

Figura 4.1: Tela do MetaMask para pedir ethers falsos



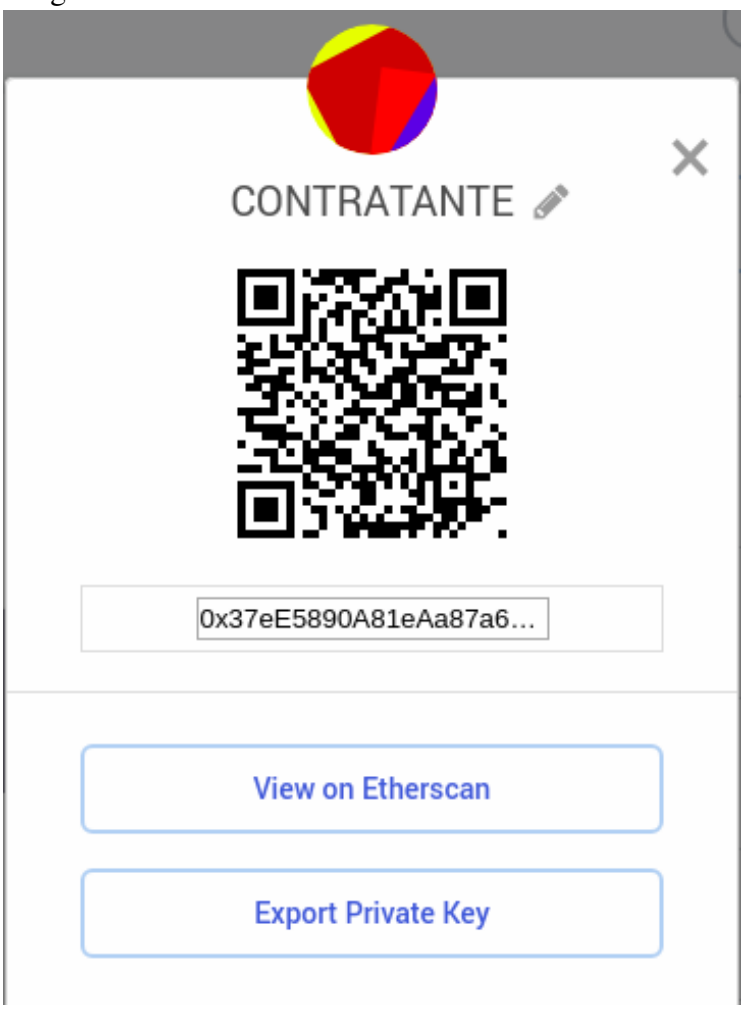
Fonte: Autor

Para entender melhor o processo de solicitar *ethers* falsos, pode-se ver a Figura 4.1, que mostra a tela após o botão de depósito, *deposit*, ter sido clicado. Deve-se então clicar na segunda opção, que diz "torneira"de teste, *Test Faucet*.

Apesar da configuração necessária para a utilização de implementações **Ethereum**, a escolha do **Remix** como ambiente de desenvolvimento tornou a configuração do ambiente simples, já que não foi necessário configurar de forma manual o **Ethereum** em si e a linguagem **Solidity**, pois o ambiente de desenvolvimento já possui *plugins* próprios para se conectar com o **Ethereum** de forma automática.

Já para a utilização e criação da aplicação contrato, foi necessário configurar uma conta no **Ethereum** via **MetaMask**, a carteira virtual, e criar uma conta no **My Ether Wallet** com a nova conta **Ethereum** criada. A criação da carteira virtual no **MetaMask** resulta também em uma "chave", a ***account token***, que pode ser vista na Figura 4.2. Ela funciona como um número de identidade, e será utilizada como identificador na criação do contrato virtual, pois é única, cada usuário possui a sua.

Figura 4.2: Carteira MetaMask com seu *account token*



Fonte: Autor

4.3 Implementação do contrato

O contrato implementado no trabalho será, na realidade, um produto mínimo viável (*Minimum Viable Product - MVP*) do que foi proposto no capítulo 3. Para isso, será utilizado um contrato de prestação de serviço com o foco em tarefas domésticas e tal implementação de código será de autoria da autora.

Baseando-se em um contrato físico de prestação de serviço (CONTRATO, 2019), algumas variáveis foram ajustadas para a utilização de um contrato inteligente, como:
Assinatura do contratante: endereço de 256 bytes, único do contratante - *account token*.
Assinatura do contratado: endereço de 256 bytes, único do contratado - *account token*.

Como testemunhas da assinatura do contrato serão os participantes da rede *blockchain* com seus respectivos endereços virtuais da rede.

As tarefas a serem cumpridas estão gravadas, em forma de comentário, de maneira imutável no início do contrato, o que torna o contrato implementado único para determinado serviço. Como mencionado antes, o contrato feito foi para prestação de serviço com foco em tarefas domésticas, como pode ser visto na Figura 4.3.

Figura 4.3: Regras do contrato implementado

```
3  // REGRAS DO CONTRATO
4
5  // As tarefas a serem feitas para que o determinado contrato seja dado como encerrado são as seguintes:
6  // 1. Retirar o pó de toda a casa, incluindo enfeites, livros, etc;
7  // 2. Varrer ou aspirar a casa;
8  // 3. Lavar e secar a louça;
9  // 4. Limpeza do banheiro;
10 // 5. Lavar e estender as roupas;
11 // 6. Limpar todos os vidros e janelas da casa;
12 // 7. Retirar o lixo.
```

Fonte: Autor

Variáveis como preço e tempo para o serviço são ajustadas no início do contrato pelo contratante e são acordadas junto com o contratado. O valor a ser pago será feito de forma integral assim que todas as tarefas forem cumpridas.

4.3.1 Conjunto de regras

O contratante só consegue abrir um contrato se sua carteira virtual não for vazia ou negativa. O mesmo vale para o contratado.

Quando o contrato for aberto, o contratante deve depositar em um depósito virtual o valor integral a ser pago pelo serviço. Tal valor só vai para a conta do contratado caso o serviço seja completo. Em casos de serviços não prestados, o valor retorna para a

conta do contratante.

Quando o contrato for aberto, o contratado deve depositar, em um depósito virtual, um quinto do valor total que será pago pelo serviço, como forma de garantia. Tal valor será retornado para o contratado quando o serviço for completo. Caso o serviço não seja cumprido, o valor se perde.

O serviço só é dado como finalizado quando ambas partes indicam que a tarefa foi cumprida. A tarefa só pode ser considerada prestada se o tempo estipulado não foi excedido. Caso o tempo seja superior ao limite, o contratado não receberá nada, mesmo que tenha feito o serviço.

Se durante o contrato for necessária alguma mudança em preço ou tempo de serviço, um novo contrato deve ser criado.

4.3.2 A implementação

O código possui uma estrutura única de contrato, com o construtor, funções e modificadores e a versão do **solidity** utilizada foi a *0.5.11*. O construtor do contrato principal, junto com as inicializações das variáveis, podem ser vistos na Figura 4.4.

No início da implementação, seis endereços são criados, linhas 25 a 31, sendo que três destes são *payable*, o que significa que podem receber *ethers*. Foi necessário criar essa quantidade de endereços, porque não é possível depositar algo para endereços que não são *payable*, apenas pode-se retirar valores destes para enviar ao contrato. O mapeamento entre o endereço comum e o endereço *payable* é feito quando o contrato é criado, linhas 47 a 49.

Além disso, na linha 33, são criados cinco estados diferentes para o contrato (Criado, Contratado, Prestado, Encerrado e Cancelado), sendo definido, assim que o contrato é criado, como valor inicial o estado "Criado", linha 45.

Durante a criação do contrato, é necessário que o contratante indique um preço para o serviço, que precisa ser em *ether*, mas o valor esperado pela interface é em *wei*, um outro tipo de criptomoeda. Para evitar, então, que o contratante tenha que calcular qual será tal valor, o preço é multiplicado por 1.000.000.000.000.000.000 (um quinquilhão) dentro do construtor, linha 41, convertendo o *wei* para *ether*.

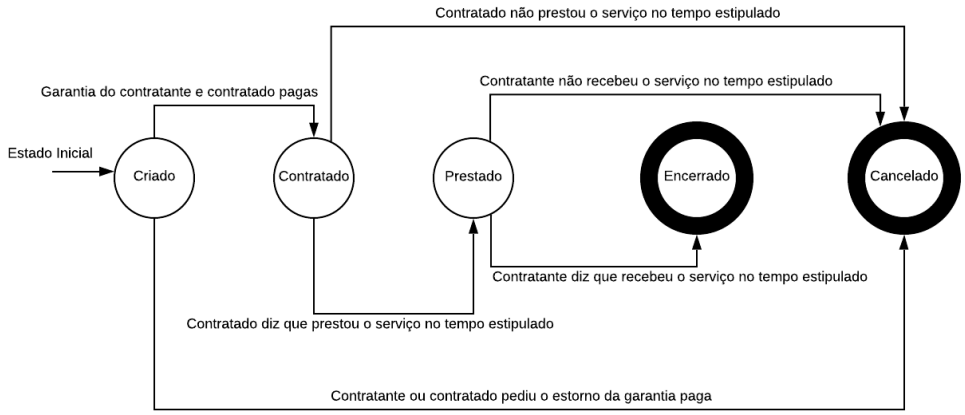
Figura 4.4: O construtor do contrato

```
15  contract MyContract {
16      uint256 public price;
17      uint8 public limiteDias;
18      uint256 deadline;
19
20      bool private garantiaContratantePaga = false;
21      bool private garantiaContratadoPaga = false;
22
23      mapping(address=>uint) balances;
24
25      address public contratante;
26      address public contratado;
27      address private contrato;
28
29      address payable payableContratante;
30      address payable payableContratado;
31      address payable payableContrato;
32
33      enum State { Criado, Contratado, Prestado, Encerrado, Cancelado }
34      State private state;
35
36      constructor(
37              uint256 _price,
38              uint8 _limiteDias,
39              address _contratado
40      ) public {
41          price = _price * 1000000000000000000;
42          limiteDias = _limiteDias;
43          contratante = msg.sender;
44          contratado = _contratado;
45          state = State.Criado;
46          contrato = address(this);
47          payableContratante = address(uint160(contratante));
48          payableContratado = address(uint160(contratado));
49          payableContrato = address(uint160(contrato));
50      }
```

Fonte: Autor

Para um melhor entendimento sobre os fluxos, a máquina de estados referente aos estados possíveis do contrato pode ser vista na Figura 4.5.

Figura 4.5: Máquina de estados dos estados do contrato



Fonte: Autor

Já a Figura 4.6, mostra algumas funções para que o contrato implementado funcione. As duas funções *modifiers*, linhas 52 e 57, são apenas para validar que aquele que está chamando a função de fato pode chamá-la. A função *deposit*, linha 62, deposita o valor total do contrato para o contratante mais a garantia paga por ele no início do contrato e só pode ser acionada se o estado do contrato é "Encerrado". Essa função é chamada de forma automática assim que o contratante confirmar o recebimento do serviço. A função *abort*, linha 67, serve para devolver ao contratante o valor pago como garantia apenas se o contrato for cancelado.

Figura 4.6: O contrato

```
52  modifier apenasContratante {
53      require(msg.sender == contratante, "Voce não é o dono deste contrato");
54      _;
55  }
56
57  modifier apenasContratado {
58      require(msg.sender == contratado, "Voce não é o prestador deste contrato");
59      _;
60  }
61
62  function deposit() private {
63      require(state == State.Encerrado);
64      payableContratado.transfer(price + price/5);
65  }
66
67  function abort() private {
68      require(state == State.Cancelado);
69      payableContratante.transfer(price);
70  }
71
72  function garantiaContratante() public apenasContratante payable {
73      require(msg.value == price,  "Preco errado");
74      balances[msg.sender] += msg.value;
75
76      garantiaContratantePaga = true;
77      if(garantiaContratadoPaga == true){
78          contratoEstabelecido();
79      }
80  }
81
82  function garantiaContratado() public apenasContratado payable {
83      require(msg.value == price/5,  "Preco errado");
84      balances[msg.sender] += msg.value;
85
86      garantiaContratadoPaga = true;
87      if(garantiaContratantePaga == true){
88          contratoEstabelecido();
89      }
90  }
91
92  function contratoEstabelecido() private {
93      require(garantiaContratadoPaga == true && garantiaContratantePaga == true, "Garantias não feitas");
94      state = State.Contratado;
95      deadline = now + (limiteDias * 1 days);
96  }
```

Fonte: Autor

As funções de pagamento de garantias podem ser vistas nas linhas 72 e 82, onde são verificados se o preço a ser pago é o correto e se quem está chamando a função tem permissão para isso. A função *contratoEstabelecido*, linha 92, é chamada de forma automática ao final do pagamentos das garantias, se ambas foram pagas, na qual o estado do contrato passa a ser "Contratado"e o limite de dias para a prestação do serviço começa a contar, linha 95.

Caso alguma das partes não pagar a garantia, o estorno é possível para a parte

que pagou e o contrato é cancelado. Tal função pode ser visualizada na Figura 4.7, linha 98.

Figura 4.7: O contrato, continuação

```
 98      function estornarGarantias() public payable  {
 99          require(state == State.Criado, "Estorno inválido");
100          if(garantiaContratadoPaga == true) {
101              payableContratado.transfer(price/5);
102              garantiaContratadoPaga = false;
103          }
104          if(garantiaContratantePaga == true) {
105              payableContratante.transfer(price);
106              garantiaContratantePaga = false;
107          }
108          state = State.Cancelado;
109      }
110
111      function servicoPrestado() public apenasContratado {
112          require(state == State.Contratado, "Contrato não válido");
113          if(now <= deadline) {
114              state = State.Prestado;
115          } else {
116              state = State.Cancelado;
117          }
118      }
119
120      function servicoRecebido() public apenasContratante {
121          require(state == State.Prestado, "Serviço ainda não prestado");
122          if(now <= deadline) {
123              state = State.Encerrado;
124              deposit();
125          } else {
126              state = State.Cancelado;
127          }
128      }
129
130      function numContrato() public view returns(address contratoAddr){
131          return address(this);
132      }
133
134      function saldoDoContrato() public view returns(uint){
135          return address(this).balance;
136      }
```

Fonte: Autor

As funções que verificam se o serviço foi prestado se chamam *servicoPrestado*, pelo lado do contratado, e *servicoRecebido*, pelo lado do contratante, linhas 111 e 120, respectivamente. Caso o contratado informe que prestou o serviço, mas no momento em que o fizer já tiver esgotado o limite de dias disponíveis, linha 113, o estado do contrato é "Cancelado"e a função *abort* pode ser chamada. Caso contrário, se o contratado informar que prestou o serviço ainda em tempo, o estado do contrato passa a ser "Prestado"e o contratante pode informar se de fato recebeu o serviço. Se sim, o estado do contrato se torna "Encerrado"e a função de depósito é chamada.

As duas últimas funções são apenas para fins de teste. A função *numContrato*,

linha 130, verifica o número de endereço atribuído ao contrato. Já na linha 134, a função *saldoDoContrato* é para confirmar que as transações (garantias, estornos e depósitos) estão funcionando da forma correta, ou seja, os valores de garantias devem entrar na conta do contrato e os valores de estorno e de depósito devem sair do mesmo e irem para contratante ou contratado.

Figura 4.8: O contrato, final

```
138     function diasRestantes() public view returns(uint){
139         if(state != State.Criado) {
140             return (deadline - now)/ 60 / 60 / 24;
141         }
142         else {
143             return 0;
144         }
145     }
146
147     function regrasContrato() public pure returns(string memory tarefas){
148         return ("As tarefas a serem feitas para que o determinado contrato seja dado como encerrado são as seguintes: 1. Retirar o pó de toda a casa,
149     }
150
151     function estadoAtualContrato() public view returns (string memory estado){
152         if(state == State.Criado){
153             return ("Criado");
154         }
155         if(state == State.Contratado){
156             return ("Contratado");
157         }
158         if(state == State.Prestado){
159             return ("Prestado");
160         }
161         if(state == State.Encerrado){
162             return ("Encerrado");
163         }
164         if(state == State.Cancelado){
165             return ("Cancelado");
166         }
167     }
168
169 }
```

Fonte: Autor

A Figura 4.8 mostra as funções de retorno ao usuário. A função *diasRestantes*, linha 138, retorna quantos dias ainda faltam para que o prazo dado expire. Já na linha 147, a função *regrasContrato* retorna as tarefas a serem cumpridas. Por último, a função *estadoAtualContrato*, na linha 151, retorna em que estado o contrato está atualmente.

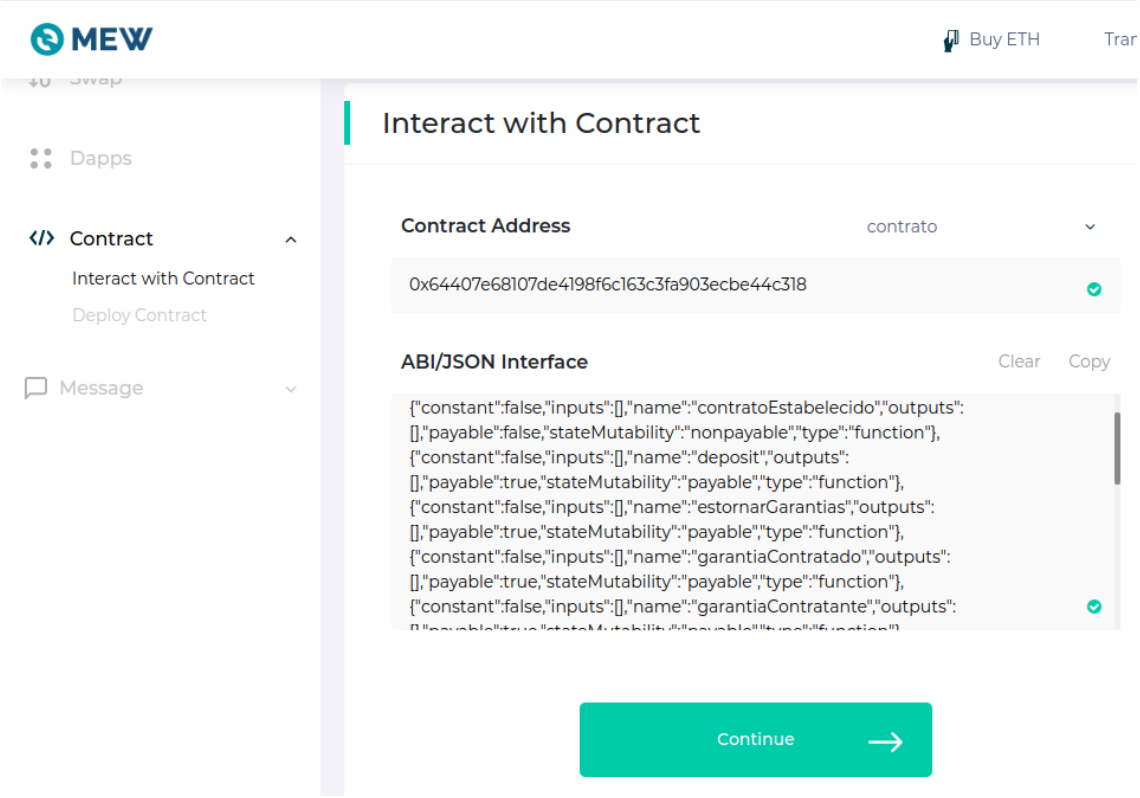
4.3.3 Criando uma instância de contrato no *My Ether Wallet*

Para criar um contrato via **My Ether Wallet**, são necessárias duas informações: o código binário e um JSON, resultantes da compilação do código fonte *Solidity*. Diversos contratos reais estão disponíveis na rede **Etherscan** (ETHERSCAN, 2019) com tais informações prontas. O contrato implementado foi feito apenas para teste, não existindo o interesse de colocá-lo na interface **Etherscan**, sendo necessário acessar o *github* da autora e ver o README (ROSA, 2019).

Com as informações, o **contratante** pode entrar na sua conta em **My Ether Wallet**, solicitar a criação do contrato, informando os dados necessários e escolhendo um nome, e em seguida interagir com o mesmo. Como podemos ver na Figura 4.9, a tela de interação mostra o endereço do contrato. Tal informação, juntamente com o JSON, é

necessária para que o **contratado** possa interagir também com o contrato recém criado.

Figura 4.9: Interação com o contrato via My Ether Wallet



Fonte: Autor

4.3.4 Teste Funcional

Para que o contrato seja dado como terminado e que a ação foi de fato cumprida, o contratado deve indicar que terminou o serviço e o contratante deve confirmar que tal afirmação é verdadeira.

Com o objetivo de evitar falsas afirmações, no início do contrato tanto o contratante quanto o contratado depositam, em um depósito virtual, uma quantia: o valor total do serviço que será prestado, no caso do contratante, e um quinto do valor do serviço no caso do contratado. Apenas quando, ou se, o serviço for prestado, o valor depositado irá para a conta do contratado, juntamente com a quantia inicial depositada.

Caso o serviço não seja prestado em um dado tempo, o valor total pago pelo contratante volta para sua conta, mas a quantia paga pelo contratado é dada como anulada.

Caso alguém queira fazer uma transação indevida, como por exemplo o contratado querer informar que o serviço foi recebido, o programa não permite e uma mensagem de erro é colocada na tela, como pode-se ver na Figura 4.10.

Figura 4.10: Permissão negada

Fonte: Autor

4.4 Avaliação Funcional

4.4.1 Metodologia

Foram criadas duas carteiras virtuais para as execuções dos testes: uma para o contratante e uma para o contratado. Serão testados quatro cenários diferentes:

1. o serviço foi prestado e o contratado recebeu o pagamento;
2. o contratado informou que o serviço foi prestado, mas o tempo já havia expirado;
3. o contratante informou que o serviço não foi prestado no tempo determinado;
4. o contratado ou o contratante não pagou a garantia.

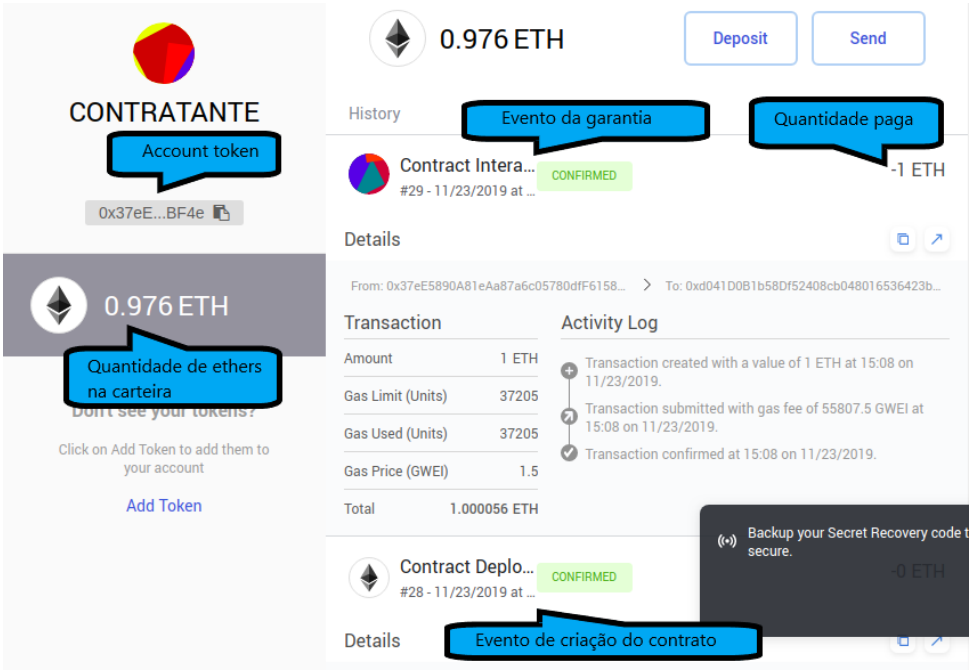
Para os testes, os contratos foram criados com o preço de um ether e um prazo para a prestação do serviço de dois dias. O critério para saber se o teste passou é verificar se o resultado esperado para cada caso de teste de fato ocorreu.

4.4.2 Cenário 1: O serviço foi prestado e o contratado recebeu o pagamento

Quando o contrato é criado pelo contratante, seu estado inicial é "Criado"e apenas quando contratante e contratado depositarem suas garantias que o contrato é dado como "Contratado", tornando essas duas ações as primeiras que devem ser feitas.

A Figura 4.11 mostra dois eventos relacionados à carteira do contratante: a criação do contrato, *Contract Deployment*, e o pagamento da garantia, *Contract Interaction*, indicando em detalhes que o valor depositado é o valor total dito na criação do contrato, no caso, um ether, juntamente com etiquetas do que significam algumas informações.

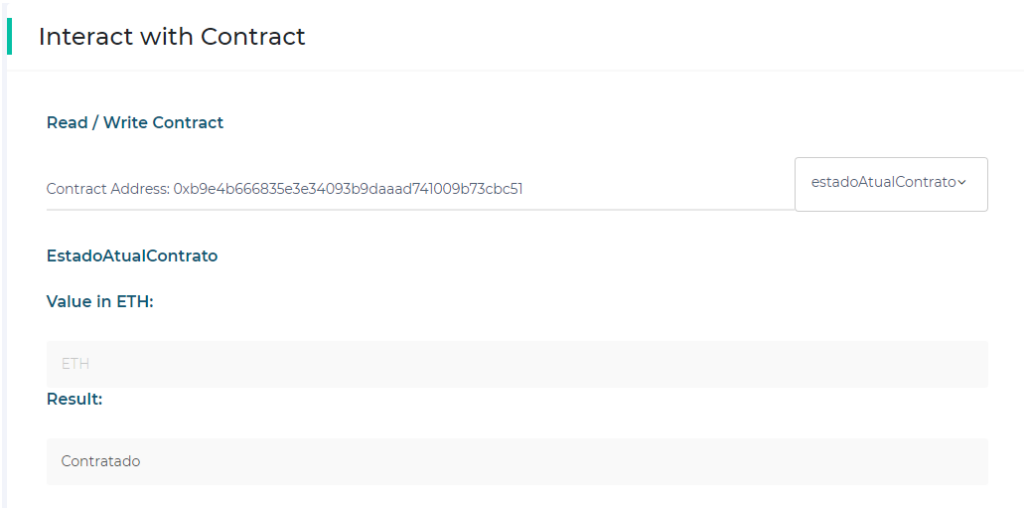
Figura 4.11: Histórico de eventos do contratante



Fonte: Autor

A ação feita pelo contratado para o depósito de sua garantia é a mesma, com exceção de que o valor a ser depositado é apenas um quinto do valor total, no exemplo utilizado seria 0,2 ether. Esse valor não é informado em nenhum lugar, sendo necessário que o contratado o calcule baseado no preço total do contrato. Assim que ambos depositarem suas garantias, o contrato é estabelecido e seu estado passa a "Contratado", conforme ilustra a Figura 4.12.

Figura 4.12: Estado do contrato como Contratado



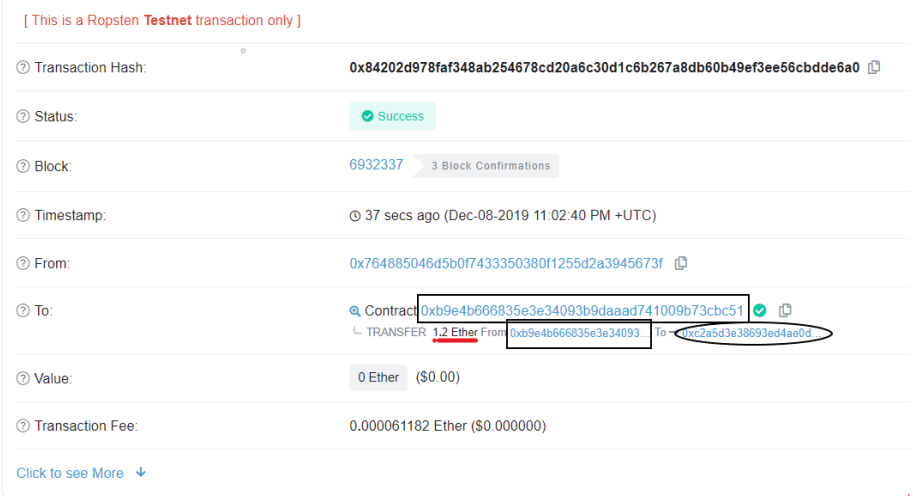
Fonte: Autor

Apenas após o contrato ter sido estabelecido que o contratado pode indicar que

prestou o serviço, ou seja, **todas** as tarefas foram feitas no tempo previsto. Para isso, ele só precisa clicar em **Serviço Prestado** na interface do **My Ether Wallet**. Apenas se essa ação for feita antes do prazo expirar que o estado do contrato se torna "Prestado"e o contratante pode confirmar a prestação do serviço. Se a ação for chamada com o prazo expirado, o contrato é automaticamente cancelado e o contratante recebe o estorno de sua parte paga. O contratado não recebe nada.

Da mesma forma que ocorre com o contratado, caso o momento em que o contratante informar que recebeu o serviço for superior ao prazo estipulado, o contrato é cancelado, mas se o tempo for menor ou igual do que o limite, o contrato passa para o estado "Encerrado"e o depósito para o contratado é feito. A Figura 4.13 mostra em detalhes a transação feita para o contratado, onde o valor a ser pago, parte sublinhada, é a soma do valor total do contrato mais a garantia paga por ele no início da contratação. A área *from* mostra a identificação (endereço) do contratante, já que a função de depósito é chamada de forma automática quando o contratante informa que recebeu o serviço e o campo *to* mostra duas identificações: a do contrato em si, que é de onde sai o pagamento (quadrados), e a do contratante, que é quem recebe (circulada).

Figura 4.13: Detalhes da transação de depósito



Fonte: Autor

4.4.3 Cenário 2: O contratado informou que o serviço foi prestado, mas o tempo já havia expirado

Caso o contratado indique que prestou o serviço em um tempo superior ao prazo limite, o estado do contrato passa a ser "Cancelado". Como o contratado já está afirmando

que não conseguiu prestar o serviço, não há necessidade de o contratante informar o mesmo, sendo possível então o cancelamento oficial do contrato (função *abort*) e o retorno da valor pago como garantia para o contratante.

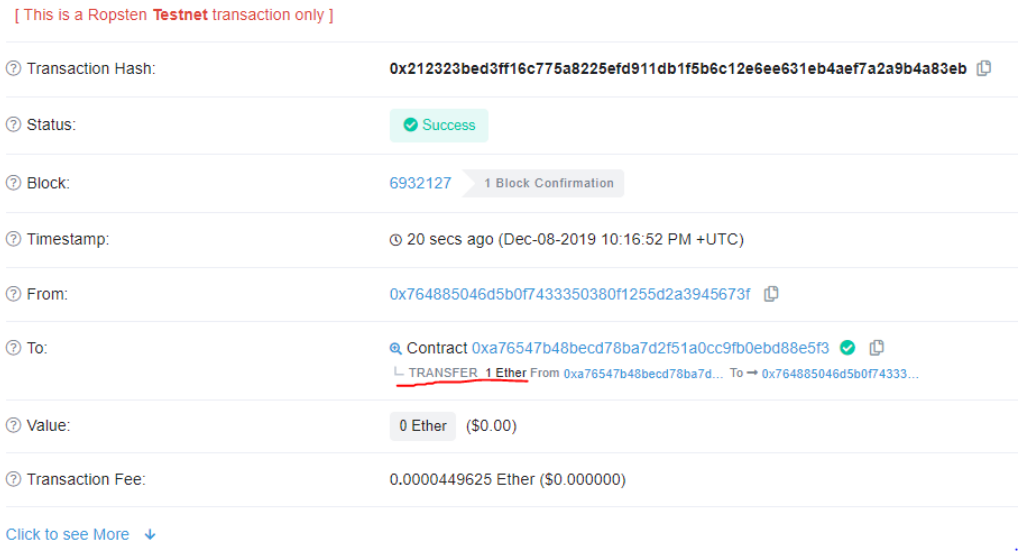
4.4.4 Cenário 3: O contratante informou que não recebeu o serviço no tempo correto

Nos casos em que o contratado afirmou que prestou o serviço, mas o contratante nega o recebimento, o contrato passa para o estado "Cancelado", e o reembolso da garantia volta para o contratante.

4.4.5 Cenário 4: O contratado ou contratante não pagou a garantia inicial

Para que o contrato seja dado como "Contratado"é necessário que ambas partes depositem uma garantia. Caso alguém não pague, é possível que aquele que pagou desista de afirmar o contrato e pegue seu reembolso. Quando o reembolso é feito, o contrato é automaticamente cancelado.

Figura 4.14: Detalhes da transação de estorno para o contratante



Fonte: Autor

A Figura 4.14 ilustra a transação de estorno entre o contrato para o contratante, onde o campo *value* é zero, já que aquele que solicitou o estorno não paga nada por isso, e campo *to* contém o endereço do contratante e o valor a ser estornado, sublinhado na Figura, no caso, 1 *ether*. Caso a situação fosse de que o contratado pagou a garantia, mas

o contratante não, o estorno se comportaria da mesma forma, mas devolvendo o valor para o contratado.

4.5 Pontos a melhorar

Em relação às **tarefas**, uma melhoria é torná-las uma das variáveis que o contratante ajusta ao criar um contrato, tornando o código implementado genérico para qualquer prestação de serviços. Já em relação ao **cancelamento do contrato**, se quem desistiu da contratação foi o contratado, o contratante poderia manter o mesmo contrato aberto mas modificar o endereço do antigo contratado para alguém que queira prestar o serviço.

5 CONCLUSÃO

Neste trabalho, foi visto como a tecnologia *blockchain* pode ser usada para a criação de um contrato inteligente, suas vantagens, desvantagens e exemplos de uso na vida cotidiana. Pode-se perceber que a tecnologia está avançando e que, assim que o número de pessoas que entendam sobre contratos virtuais aumentar, os contratos físicos podem ser facilmente extintos.

Observa-se que os benefícios desse tipo de contrato tanto para contratado como contratante são: **segurança jurídica**, já que diversas empresas estão se adaptando a essa nova realidade de contratação e assinaturas eletrônicas; **economia de tempo e dinheiro**, visto que que não há o envolvimento de terceiros e o processo é rápido; **transações mais limpas e transparentes**, pois todos os registros estão na rede *blockchain*; **baixo índice de fraudes**, porque uma vez definidos os requisitos, não há a possibilidade de nenhum tipo de alteração; **rapidez na tramitação burocrática**, pois os papéis e toda a burocracia normal de um contrato físico deixam de existir; **opção mais sustentável** indo de acordo com a responsabilidade social de cada empresa; **velocidade nos negócios**, já que reduz o tempo de negociação, e há possibilidade de novos fechamentos com rapidez; **maior interação tecnológica**; **armazenamento seguro**, visto que o contrato está na rede *blockchain*; e por fim **documentos disponíveis a qualquer tempo** (ABRIL BRANDED CONTENT, 2017).

O que pode-se concluir deste trabalho é que implementar um contrato inteligente não é simples, pois envolve encontrar as melhores plataformas, entender a linguagem utilizada e, o mais difícil, subir o código para a rede *blockchain*, mas utilizá-lo custa menos tempo e dinheiro, sendo necessário apenas ensinar às partes quais as informações necessárias para utilizá-lo.

Na realização desde trabalho foram encontradas várias dificuldades, indo desde a inicialização do ambiente *blockchain*, já que é necessário importar todos os blocos da cadeia para o computador, o que leva algumas horas, até o funcionamento do código implementado.

Uma das maiores dificuldades, foi encontrar como obter *ethers* falsos para poder criar o contrato e ajustar o código para que o mesmo buscasse de fato os valores nas carteiras do **MetaMask**. Além disso, dependendo da transação a ser executada, o aplicativo do **MetaMask** trancava o computador inteiro, sendo necessário reinicializá-lo. Outra dificuldade foi encontrar um ambiente que traduzisse o contrato para uma interface em

que o usuário pudesse mexer sem ter acesso ao código.

Para trabalhos futuros pode-se incluir as melhorias apontadas na seção 4.5, juntamente com a realização de doações para carteiras de entidades que prestam caridade com os valores que não seriam recuperáveis do contrato, no caso as garantias não recuperáveis do contratado. Além disso, pode-se avaliar de forma mais profunda os custos entre um contrato físico e um contrato virtual, focando nas questões econômicas e não em dificuldades de uso ou implementação.

REFERÊNCIAS

ABRIL BRANDED CONTENT. **5 benefícios da assinatura eletrônica para as empresas**. 2017. <https://exame.abril.com.br/tecnologia/5-beneficios-da-assinatura-eletronica-para-as-empresas/>. Acessado em 01 de dezembro de 2019.

AGATETEPê. **Construa seu primeiro Ethereum Smart Contract com solidez – Tutorial**. 2018. <https://www.agatetepe.com.br/construa-seu-primeiro-ethereum-smart-contract-com-solidez-tutorial/?fbclid=IwAR0GzxzZktkk_yodNaKCguOhR5NAjgtLw2XOxi3DJMBluxSt9090waF4gBQ>. Acessado em 26 de outubro de 2019.

BITCOIN. 2009. <https://bitcoin.org/en/>. Acessado em 24 de novembro de 2019.

BITCOIN WIKI. **Proof of Work**. 2019. <https://en.bitcoin.it/wiki/Proof_of_work>. Acessado em 28 de novembro de 2019.

BITDEGREE. **What Is a Smart Contract and How Does it Work?** 2019. <https://www.bitdegree.org/tutorials/what-is-a-smart-contract>. Acessado em 27 de novembro de 2019.

BLOCKCHAINTECHNOLOGIES.COM. **Blockchain Applications in Internet of Things (IoT)**. 2019. <https://www.blockchaintechnologies.com/applications/internet-of-things-iot/>. Acessado em 24 de novembro de 2019.

BLOCKCHAINTECHNOLOGIES.COM. **Blockchain Applications in Money**. 2019. <https://www.blockchaintechnologies.com/applications/money/>. Acessado em 24 de novembro de 2019.

CARDOSO, B. **Contratos inteligentes: descubra o que são e como funcionam**. 2018. <https://brunonc.jusbrasil.com.br/artigos/569694569/contratos-inteligentes-descubra-o-que-sao-e-como-funcionam>. Acessado em 05 de setembro de 2019.

CHAIN core. 2016. <http://fedchains.com/core/>. Acessado em 24 de novembro de 2019.

CONTRATO. 2019. <http://www.lex.com.br/contrato_1127004_CONTRATO_DE_PRESTACAO_DE_SERVICOS_AUTONOMO.aspx>. Acessado em 30 de outubro de 2019.

CORDA. 2019. <https://www.corda.net/>. Acessado em 24 de novembro de 2019.

CROSBY, M. et al. **BlockChain Technology - Beyond Bitcoin**. [S.l.], 2015.

CRYPTO BEGINNERS. **Blockchain-in-1000-words**. 2019. <https://cryptobeginners.info/blog/what-is-blockchain/>. Acessado em 15 de setembro de 2019.

DANIEL. **Cryptographic Hash Functions Explained: A Beginner's Guide**. 2018. <https://komodoplatform.com/cryptographic-hash-function/>. Acessado em 05 de novembro de 2019.

DOB, D. **Permissioned-Permissionless-Blockchains: Understanding the Differences**. 2018. <https://blockonomi.com/permissioned-vs-permissionless-blockchains/>. Acessado em 10 de setembro de 2019.

ENCRYPGEN. 2018. <https://encrypgen.com/>. Acessado em 27 de novembro de 2019.

ETHEREUM. 2019. <https://www.ethereum.org/>. Acessado em 30 de outubro de 2019.

ETHERSCAN. 2019. <https://etherscan.io/>. Acessado em 09 de novembro de 2019.

FAUSTINO, D. **Blockchain e suas Aplicações**. 2019. <https://esr.rnp.br/webinar/blockchain-aplicacoes>. Acessado em 16 de maio de 2019.

FILHO, A. M. D. S. **Artigo Engenharia de Software 3 - Requisitos Não Funcionais**. 2008. <https://www.devmedia.com.br/artigo-engenharia-de-software-3-requisitos-nao-funcionais/9525>. Acessado em 07 de dezembro de 2019.

FOLLOWMYVOTE. 2019. <https://followmyvote.com/>. Acessado em 27 de novembro de 2019.

FRANKENFIELD, J. **Proof of Stake (PoS)**. 2019. <https://www.investopedia.com/terms/p/proof-stake-pos.asp>. Acessado em 24 de novembro de 2019.

GAMA, A. **O que é JSON**. 2011. <https://www.devmedia.com.br/o-que-e-json/23166>. Acessado em 21 de dezembro de 2019.

GANDHI, R.; RAMASASTRI, D. A. S. **Applications of Blockchain technology to banking and financial sector in India**. [S.l.], 2017.

GRYBNIAK, S. **Smart-Contracts-Financial-Blockchain-Systems**. 2017. <https://hackernoon.com/advantages-and-disadvantages-of-smart-contracts-in-financial-blockchain-systems-3a443145ae1c>. Acessado em 10 de setembro de 2019.

GUIA DO BITCOIN. **Hash-Functions**. 2017. <https://guiadobitcoin.com.br/se-voce-entender-a-funcao-da-hash-voce-entendera-a-blockchain/>. Acessado em 09 de setembro de 2019.

HOODA, P. **Raft Consensus Algorithm**. 2019. <https://www.geeksforgeeks.org/raft-consensus-algorithm/>. Acessado em 24 de novembro de 2019.

HYPERLEDGER. 2018. <https://www.hyperledger.org/>. Acessado em 24 de novembro de 2019.

KONSTANTOPOULOS, G. **Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance**. 2017. <https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419>. Acessado em 24 de novembro de 2019.

LEX MAGISTER. **CONTRATO DE PRESTAÇÃO DE SERVIÇOS (AUTÔNOMO)**. 2019. <http://www.lex.com.br/contrato_1127004_CONTRATO_DE_PRESTACAO_DE_SERVICOS_AUTONOMO.aspx>. Acessado em 21 de novembro de 2019.

LIVE COINS. **Algoritmos de Consenso**. 2019. <https://livecoins.com.br/algoritmos-de-consenso/>. Acessado em 24 de novembro de 2019.

MARINHO, J. J. **Análise e levantamento de requisitos em histórias em quadrinhos – PARTE 2: A obscura diferença entre requisitos funcionais e requisitos não funcionais**. 2015. <https://www.tiespecialistas.com.br/analise-e-levantamento-de-requisitos-em-historias-em-quadrinhos-parte-2-obscura-diferenca-entre-requis>. Acessado em 07 de dezembro de 2019.

METAMASK. 2019. <https://metamask.io/>. Acessado em 04 de novembro de 2019.

MYETHERWALLET. 2019. <https://www.myetherwallet.com/>. Acessado em 23 de novembro de 2019.

PRADO, J. **O que é blockchain? [indo além do bitcoin]**. 2017. <https://tecnoblog.net/227293/como-funciona-blockchain-bitcoin/>. Acessado em 02 de setembro de 2019.

PRATAP, M. **Everything You Need to Know About Smart Contracts: A Beginner's Guide**. 2018. <https://hackernoon.com/everything-you-need-to-know-about-smart-contracts-a-beginners-guide-c13cc138378a>. Acessado em 27 de novembro de 2019.

PRISCO, G. **Chainalysis and Wave Showcase Blockchain Fintech Products at New York Barclays Accelerator, Sign Deal with Barclays**. 2015. <https://bitcoinmagazine.com/articles/chainalysis-and-wave-showcase-blockchain-fintech-products-at-new-york-barclays-accelerator-sign-deal-> Acessado em 27 de novembro de 2019.

R3. 2019. <https://www.r3.com/>. Acessado em 24 de novembro de 2019.

REMIX. 2019. <https://github.com/ethereum/remix-ide>. Acessado em 26 de outubro de 2019.

REMIX-IDE. 2019. <https://remix.ethereum.org/#optimize=false&evmVersion=null&version=soljson-v0.5.11+commit.c082d0b4.js>. Acessado em 04 de novembro de 2019.

RIBEIRO, L. **O que é UML e Diagramas de Caso de Uso: Introdução Prática à UML**. 2012. <https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408>. Acessado em 07 de dezembro de 2019.

ROSA, V. **Github**. 2019. <https://github.com/vitoriamr/SmartContract/tree/master>. Acessado em 23 de novembro de 2019.

ROSIC, A. **17 Blockchain Applications That Are Transforming Society**. 2016. <https://blockgeeks.com/guides/blockchain-applications/>. Acessado em 24 de novembro de 2019.

ROSIC, A. **Smart Contracts: The Blockchain Technology That Will Replace Lawyers**. 2016. <https://blockgeeks.com/guides/smart-contracts/>. Acessado em 05 de setembro de 2019.

ROSIC, A. **What is Ethereum?** 2016. <https://blockgeeks.com/guides/ethereum/>. Acessado em 10 de setembro de 2019.

SOLIDITY. 2016. <https://solidity.readthedocs.io/en/v0.5.12/>. Acessado em 30 de outubro de 2019.

STACKEXCHANGE. **Ropsten Test Network**. 2017. <https://ethereum.stackexchange.com/questions/13534/what-is-actually-ropsten-what-is-a-new-network>. Acessado em 01 de dezembro de 2019.

TAKYAR, A. **Blockchain IoT Use Cases**. 2019. <https://www.leewayhertz.com/blockchain-iot-use-cases-real-world-products/>. Acessado em 04 de dezembro de 2019.

THE-LINUX-FOUNDATION. 2019. <https://www.linuxfoundation.org/>. Acessado em 24 de novembro de 2019.

VOSHMGIR, S. **What is Blockchain?** 2019. <https://blockchainhub.net/blockchain-intro/>. Acessado em 24 de novembro de 2019.

YAGA, D. et al. **NISTIR 8202-Blockchain Technology Overview**. [S.l.], 2018.